Calculator for Solving Two-Person Zero-Sum Games

Solonas Argyrides

BSc (Hons) Computer Science

2005
Title: “Calculator for solving Two-Person Zero-Sum games”

Submitted by Solonas Argyrides

COPYRIGHT

Attention is drawn to the fact that copyright of this thesis rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinance/#intelprop).

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author.

DECLARATION

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution or learning. Except where specifically acknowledged, it is the work of the author.

Signed: ……………………

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.
Abstract

Game theory aroused much interest due to the mathematical properties and the number of applications to economics, political and social problems. Game theory can be used in complex situations that involve more factors than somebody’s choice and chance as a tool for making decisions. In this dissertation, the type of games considered is the two person zero sum. A program has been developed for solving any given matrix game. The program, which works as a calculator, returns the optimal strategies for two players and the value of a game. The method that is used for solving the matrix games is by using linear algebra and the simplex method. This program can be used as a learning program for teaching game theory applications together with simplex method process. Additional to that, it can be used as a calculator for finding quick solutions as well as for checking all the steps of simplex method since this method is a complicated and a time consuming method when performed by hand. The design and the implementation of this program are presented within. The testing of the system through several methods then follows. The dissertation is concluded with a discussion on the program produced together with further additions that could be made. An appendix is also included that contains a reference manual for the program.
I would like to thank my project supervisor Dr Nicolai Vorobjov, for his ongoing support, help and invaluable advice throughout this project. I would also like to thank my family back in Cyprus for the psychological support that they gave me during this difficult for me time.
# Table Of Contents

<table>
<thead>
<tr>
<th>Chapters</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Literature Review</td>
</tr>
<tr>
<td>2.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2.2</td>
<td>Discussion and background theory</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Definition of a game</td>
</tr>
<tr>
<td>2.2.2</td>
<td>General Definitions</td>
</tr>
<tr>
<td>2.2.3</td>
<td>The Two-Person, Zero-Sum Games</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Pure Strategies</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Mixed Strategies</td>
</tr>
<tr>
<td>2.2.6</td>
<td>The Minimax Theorem</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Optimal Strategies</td>
</tr>
<tr>
<td>2.2.8</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>2.2.9</td>
<td>Simplex method</td>
</tr>
<tr>
<td>2.3</td>
<td>Summary</td>
</tr>
<tr>
<td>3</td>
<td>Requirements Analysis &amp; Requirements Specification</td>
</tr>
<tr>
<td>3.1</td>
<td>Requirements Analysis</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Scope of the product</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Purpose of the product</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Requirements Elicitation and Analysis</td>
</tr>
<tr>
<td>3.2</td>
<td>Requirements Specification</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Requirements drawn from Literature Review</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Functional Requirements</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Non-Functional Requirements</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Hardware Requirements</td>
</tr>
<tr>
<td>3.2.5</td>
<td>User Requirements</td>
</tr>
<tr>
<td>3.3</td>
<td>Requirements Validation</td>
</tr>
<tr>
<td>4</td>
<td>Design</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.2</td>
<td>Module Overview</td>
</tr>
<tr>
<td>4.3</td>
<td>Data Structures</td>
</tr>
<tr>
<td>4.4</td>
<td>Mode Selection</td>
</tr>
<tr>
<td>4.5</td>
<td>Minmax Maxmin Method</td>
</tr>
<tr>
<td>4.6</td>
<td>Simplex Method</td>
</tr>
<tr>
<td>4.7</td>
<td>User Interface Design</td>
</tr>
<tr>
<td>4.7.1</td>
<td>User input</td>
</tr>
<tr>
<td>4.7.2</td>
<td>User feedback</td>
</tr>
<tr>
<td>4.8</td>
<td>Error Handling</td>
</tr>
</tbody>
</table>
5 Detailed Design and Implementation

5.1 High Level Implementation Decisions

5.1.1 Platform

5.1.2 Implementation Language

5.1.3 Development Environment

5.2 Overall System architecture

5.3 Mode method

5.4 Data Input methods

5.4.1 Game Matrix Dimensions

5.4.2 Game Matrix Elements

5.5 Truth checking methods

5.6 Display functions

5.7 Minimax Maximin method

5.8 Simplex method

6 Testing and Evaluation

6.1 Unit testing

6.2 Verification and Validation

6.3 Defect Testing

6.3.1 Black Box Testing

6.3.2 Structural Testing

6.4 Integration Testing

7 Conclusion

Bibliography

APPENDIX A

A1 Evaluation of existing systems

A2 Questionnaire for potential users

A2 Use Cases

APPENDIX B

B1 Inspection and Evaluation testing

B2 Defect Testing

B3 Book Testing

APPENDIX C

C1 Reference Manual

APPENDIX D

D1 The Code
Chapter 1

Introduction

To obtain a real understanding of the problem of exchange by studying it from an altogether different angle; this is, from the perspective of a “game of strategy.”

John von Neuman
The theory of Games and Economic Behaviour
1 Introduction

John Von Neumann in 1928 laid the foundations of game theory by proving the basic minimax theorem. The field was established in 1944 when he published the ‘Theory of games and Economic Behaviour’, where it was shown that social events can be described better by some models of the appropriate games of strategy. John Von Neumann was describing situations that were competitive and the loss of one player was the gain of another, therefore the name zero sum was given for these situations. After John Nash’s improvements, game theory can now describe and find optimum strategies for a number of different types of games, which are not limited in the zero sum type constraint. That is the reason why game theory is of great importance in the social sciences field.

The purpose of this dissertation is to develop a program that is used for solving any given matrix game that is of the two person zero sum type. The program is going to be able to find the optimal strategies for the two players and the value of the game given. The program will be implemented in finding the solutions in two ways. One way is the minimax theorem and the other way is with the use of linear algebra and the simplex method. Furthermore, another program’s aim will be to provide all steps of the simplex method when performed, so as to be feasible for people to use it when solving linear program problems since it is a difficult and time consuming method.

The material to be presented can be read from anybody that is interested in the field even if his/her background is not strong in mathematics. The literature review, chapter number 2, provides an in-depth description for understanding the mathematics needed for the purpose of solving game matrices. Furthermore, definitions are provided for the language and symbols that are used. Additional information and descriptions about the algorithms used are provided, as well as step-by-step explanations.

Chapter number 3 describes all the methods that were used for gathering the requirements for the program to be developed. Chapter number 4 describes all the design methods that were used for the development of the final system. Also a detailed description of the simplex algorithm can be found in chapter 4, with a step-by-step guidance for solving with this method.

A complete description of how the methods that were implemented and why they were implemented in the way that they did, can be found in the chapter number 5, which is the detailed design and implementation chapter. The testing follows in chapter number 6, describing all the method used for the purpose of testing together with explanations on what aim had each test that was used.

Conclusions of this dissertation can be found at the last section, chapter number 7, together with suggestions for further improvements. At the end of this document, an enclosed CD can be found that includes the program developed.
Chapter 2

Literature Review

This chapter’s aim is to introduce the mathematics needed for understanding how game theory problems can be solved. Also, methods and algorithm needed are also mentioned and explained…
2 Literature Review

2.1 Introduction

Defining what game theory is can be very general, because it is a broad topic. There are many theories for games but there is not one theory of games. The theory is how someone should make certain decisions, and how he makes these decisions. When you know what you want to achieve, the set of actions which you can chose from and the consequence of each action, it should be fairly easy to reach your goal. But decision-making is harder when chance is involved. “Game theory was designed as a tool for making decisions in complex situations that many factors are operating, not only luck and someone’s choice.”\(^1\)

Parlor games like chess, bridge, roulette and many others, all begin from a start point and after a sequence of moves that the players choose from, among several possibilities, “strategies”, they all finish due to the rules of each individual game has. (In game theory, strategy is what a player will do, what plan he is going to follow under all possible circumstances.) Rules are for example whether communication is allowed or not between players, binding agreements, what information they have, if they have the ability to share the “payoff” and so on. Also, since the game is finite, the players have a finite number of strategies as well. The number of players involved is really important as well as how much the players interests clash or overlap. When the game finishes, there is some “payoff” for the player that wins. This payoff can either be satisfaction, prestige or even money.\(^2\)

Parlor games can be based entirely on luck, like roulette, or they can be entirely based on skills, like chess. Also there are some others, like bridge, where luck plays an important role but still skills are also needed.

Furthermore, in games like chess, checkers, tick-tack-toe, both players know all the moves that have been made at any time. These kinds of games are called games of perfect information, and they offer some conceptual problems. In games like bridge and poker however, the players’ knowledge is usually imperfect because the players are kept in the dark. That is why these kinds of games are more complex. For example the kids’ game of matching pennies, although it is simple, has an extra dimension of complexity due to the fact that players do not know the opponents decision.

In general, a game should be concerned with three elements. Move changing either by chance or by personal, the knowledge that one might have or not have, and the payoff function.\(^3\)

The mathematical approach that it will be followed to explain how everything works is based on the approach that Guillermo Owen followed in his book Game Theory, 2\(^{\text{nd}}\) edition, in 1982.

---

2 Owen Guillermo (1968), 'Game Theory' 1\(^{\text{st}}\) ed. p.2, W. B. Saunders.
2.2 Discussion and background theory

2.2.1 Definition of a game

We can represent a game by a ‘game tree’, also called a ‘topological tree’. It includes vertices, which are a collection of nodes, which are connected by arcs, straight lines. That is a connected figure that does not include simple closed curves. So, if we have vertices A and B, there is a sequence of arcs and nodes that is unique and joins vertex A and vertex B.

If in a topological tree there is a distinguished vertex A, and the sequence of arcs joining A to C passes through B then we say that vertex ‘C follows B’. We say that C follows B ‘immediately’ if B is just before C and an arc exists that joins B to C. Also we say that a vertex is terminal if no other vertex follows after it.

2.2.2 General Definitions

A) A topological tree $G$, of a game $G$, with a distinguished vertex A is called the starting point of $G$. Since a game always has a starting point.

B) The payoff function is a function that assigns an n-vector to each terminal vertex of $G$. That gives the payoff when the game is finished. Mathematically is represented by: $\pi(\sigma_1, \sigma_2, ..., \sigma_n) = (\pi_1(\sigma_1, ..., \sigma_n), \pi_2(\sigma_2), ..., \pi_n(\sigma_1, ..., \sigma_n))$ where player $i$ is using strategy $\sigma_i \in \sum_i i$.

C) A partition of the non-terminal vertices of $G$ into n+1 sets $S_0$, $S_1$, ..., $S_n$, called the player sets. That is a partition of the chance moves $S_0$ and the player (n) moves into $S_1$, ..., $S_n$.

D) A probability distribution is defined at each vertex of $S_0$, among the immediate followers of this vertex. That is to define a random scheme at every chance move.

E) For each $i = 1, ..., n$ a sub partition of $S_i$ into subsets $S_i^j$, called information sets, such that two vertices in the same information set have the same number of immediate followers and no vertex can follow another vertex in the same information set. This means players moves are divided into information sets. The players know the information set are following but the vertex of that information set is unknown to them though.

F) For each information set $S_i^j$, an index set, an index set $I_i^j$, together with a 1-1 mapping of the set $I_i^j$ onto the set of immediate followers of each vertex of $S_i^j$.

---

G) A strategy for a player $i$ is a function that assigns to every information sets $S^i_j$ of player $i$, one of the arcs that come after a representative vertex of $S^i_j$.

H) We define a set with all strategies $\sum_i$, for the player $i$.

Let's take for example a simple game of Heads and Tails. Player A chooses first. Then Player B chooses without knowing what Player A has chosen. If they both have chosen the same, then Player A wins a pound from Player B. But if the selection of pennies side was different then Player B wins a pound from Player A.

The topological tree for this simple game is shown below. At each terminal vertex there is a vector that shows the payoff function. At the other vertices before the terminals, the player whose turn it is to play is shown. Moves that belong to the same information set are shown by the red line, which encloses them.

In a game, there sometimes exists a point, one or more, that is called equilibrium point. That point(s), gives the players a stable outcome associated with a pair of strategies. This point is always considered to be stable due to the fact that if a player changes his strategy by his own will and the opponent does not then the former is hurt by that change. If equilibrium points exist then in general we can find them easily.

**Definition:** The point $(X_0, Y_0)$ is called equilibrium situation if for any $X, Y$:

$$H(X, Y) \leq H(X_0, Y_0) \leq H(X_0, Y)$$

For $X$ to be the optimal strategy for Player A, and $Y$ to be the optimal strategy for Player B.

In a game $\Gamma$ we will call normal form of the game a matrix that its rows and columns represent the strategies of Players A and B respectively. For example the Heads and Tails game described above has the normal form:

<table>
<thead>
<tr>
<th></th>
<th>Heads</th>
<th>Tails</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heads</td>
<td>1,−1</td>
<td>−1,1</td>
</tr>
<tr>
<td>Tails</td>
<td>−1,1</td>
<td>1,−1</td>
</tr>
</tbody>
</table>
Where Player A has the strategies Heads $[1, -1, -1, 1]$ and Tales $[-1, 1, -1]$ and Player B has the strategies Heads $[1, -1, -1]$ and Tales $[-1, 1]$.

This particular game though does not have any equilibrium points, due to the fact that players try to outguess each other’s strategies keeping their own strategy secret. In games with perfect, complete information though equilibrium points and strategies exist.

### 2.2.3 The Two-Person, Zero-Sum Games

These are games that played by two players or two teams, against each other. That is why they also called strictly competitive games. They are referred to as zero sum because they work as a closed system. That is everything that Player A will win is exactly the same amount that Player B will lose. Most parlor games are of this type. Since in this type of games whoever wins the other one loses, there is no need, and there is no point for negotiation between the players, or teams.

**Definition:** If and only if at every terminal vertex the payoff function $(p_1, \ldots, p_n)$ satisfies $\sum_{i=1}^{n} p_{i=0}$ then a game $\Gamma$ is said to be of zero sum.

Since the payoff function that assigns a vector to the winner at the terminal vertex is going to be the same number that the other player lost, so only the first component of the vector is given. The other component is the same as the first but negative.

For the two-person, zero-sum type games, when equilibrium points exist, and if equilibrium points are more than one, then all of them will have the same payoff. Also if equilibrium points exist then they are called the solution of the game or the value of the game.

Since in these types of games the payoff function is reverse analogical, if equilibrium points exist then an important theorem comes from this condition. It states that if $(\sigma_1, \sigma_2)$ and $(\tau_1, \tau_2)$ are two equilibrium points then:

- a) $(\sigma_1, \tau_2)$ and $(\tau_1, \sigma_2)$ are also equilibrium points,
- b) $\pi(\sigma_1, \sigma_2) = \pi(\tau_1, \tau_2) = \pi(\sigma_1, \tau_2) = \pi(\tau_1, \sigma_2)$.

This theorem though is not true for different type of games.

Here, the normal form consists of a simpler matrix since we only need the first component of the vector. The payoff is going to be the element $\sigma_{ij}$ of this matrix.

---

5 Owen Guillermo (1982), 'Game Theory' 2nd ed. p.10 and also more detail can be found in the book of Binmore, Ken. Fun and Games - A Text on Game Theory, 1992-p.131-133.
when Player A chooses his ith row strategy and when Player B chooses his jth column strategy.

For example, let's consider the game matrix below:

\[
M = \begin{bmatrix}
1 & 2 & 3 \\
6 & 5 & 6 \\
3 & 2 & 1
\end{bmatrix}
\]

If Player A chooses the first row as his strategy, and Player B chooses as his strategy the third column of the matrix, then the payoff\(^6\) function will be 3. Also in this game there is an equilibrium strategy. The \(\alpha_{3,2}\) element of the game matrix above is the biggest in its column and at the same time the smallest in its row.

That is: \(\alpha_{Aj} \leq \alpha_{ij} \leq \alpha_{jB}\)

This element is also called a saddle point, when it exists. In the Head and Tails game described above, saddle points did not exist. The importance of saddle points is that when they do exist there is no need for the Players to keep their strategies secret. They just stick to the specific strategy. Even if one of the Players outguesses what the other Players strategy is, there is no point in not following it since both players want to maximize their payoff and minimize the payoff of their opponent. In this type of games though, since the vector consists only of the first element, Player B will try to do that by minimising the payoff function since his outcome will be more when the outcome of Player A is going to be negative.

### 2.2.4 Pure Strategies

Pure strategy \(i\) of a Player A is a mixed strategy having components equal to 0 except from the \(i\)th place:

\[
\left(0 \ldots 0 1 0 \ldots 0\right)
\]

Let us suppose that the numbers within the matrix represent the number of pounds. If Player A tries to select a strategy that will give him the biggest payoff, which here is the first row, Player B will understand it and vice versa. This is because we consider that both players are fairly clever.

The proper thing to do, is for Player A to select the strategy \((8 \quad 4)\), for the reason that in this strategy A will win at least 4 pounds whereas the other strategy will give

---

\(^6\) More on payoff can be found: Binmore, Ken. Fun and Games - A Text on Game Theory, 1992. p.149.
Player A, only 2 pounds payoff. We will call this ‘gain-floor’ since it gives Player A the certain win of at least 4 pounds. The gain-floor is denoted by:

$$v^* = \max_i \{ \min_j \ c_{ij} \}$$

For Player B, the choice is to select the second strategy. The conditions of the choice will be the same. This is called the ‘loss-ceiling’ and in this case will be three pounds. The ‘loss-ceiling’ is denoted by:

$$v^{\circ} = \min_j \{ \max_i \ c_{ij} \}$$

Also $v^{\circ} \leq v^*$. When the two are equal then saddle points exists. If not, then we know that the game does not have any saddle points. If a game does not have any saddle points, both Players A and B must keep their strategies secret in order to do better. If they do not manage to keep their strategies secret then the best they can hope for, is to get either the ‘gain-floor’ or the ‘loss-ceiling’ respectively.

### 2.2.5 Mixed Strategies

It is always better for a Player to keep his strategy secret, but as in the previous example though, a clever Player can reconstruct his opponents strategy by thinking “if I was my opponent, what would I select…?” At the same time if players choose strategies irrationally though, then there is no point for analysing it. The solution is that the players should choose their strategies randomly but under a rational randomization scheme.

In general, a mixed strategy of a player of a no cooperative game is a mixed strategy that gives the probability weight to all pure strategies available to the players. Players’ mixed strategy is a probability distribution on the set of the players’ pure strategies. When a player has a finite number of pure strategies, $m$, a mixed strategy is a vector with dimension $m$, such that:

$$x = (x_1, \ldots, x_m), \text{ Which satisfies } x_i \geq 0 \text{ and }$$

$$\sum_{i=1}^m x_i = 1.$$  

In a game $\Gamma$ the payoff when Player A chooses a mixed strategy $x$ and Player B chooses a mixed strategy $y$, will be:

$$\Gamma (x, y) = \sum_{i=1}^m \sum_{j=1}^n x_i c_{ij} y_j.$$  

In a matrix notation that would be represented as:

$$\Gamma (x, y) = x^T \Gamma y,$$
Now if Player B discovers Players A strategy, then Player B will choose $y$ so as to minimize $\Gamma(x, y)$. That is because if Player A uses strategy $x$ his gain-floor will be $v(x) = \min_{y \in Y} x\Gamma y^T$, and $x\Gamma y^T$ is the average of the payoffs for Player A when uses strategy $x$ against Player’s B pure strategies. Therefore, a pure strategy $j$ will be used, to get the minimum by: $v(x) = \min_{y \in Y} x\Gamma y^T$, with $x\Gamma_j$ to be the $j$th column of the game matrix $\Gamma$.

So Player A should choose strategy $x$ in order to make $v(x)$ maximum, and therefore to obtain: $v_A = \max_{x \in X} \min_{j \in J} x\Gamma_j$. This $x$ strategy for Player A is called the maximin strategy. The number $v_A$ that we get is called the value of the game for player A.

The same concept stands for Player B as well, meaning that to obtain the ‘loss-ceiling’ $v(y) = \max_{y \in Y} \min_{i \in I} y\Gamma_i$, he has to choose strategy $y$. Here, the $\Gamma_y$, is the $i$th row of the matrix $\Gamma$. Furthermore, as with Player A, Player B will choose $y$ in order to obtain: $v_B = \min_{y \in Y} \max_{i \in I} \Gamma_i y^T$.

This $y$ strategy for Player B is called the minimax strategy. The number $v_B$ that we get is called the value of the game for player B.

So, in conclusion to that, if a Player chooses a mixed strategy, the payoff will be the same whatever strategy his opponent will choose. That means both players will get the payoff on average.

2.2.6 The Minimax Theorem

The Minimax theorem, which was given by von Neumann and proved by him in 1928, is one of the fundamental and most important theorems of game theory. It states that we can assign a value to all finite two-person zero-sum games. This value, $V$, is the average amount that Player A is able to win from player B when both players play sensibly. This is true because of the following reasons:

a) There is a strategy that Player A can follow that will protect his payoff return of the value $V$. Player B can not do anything to prevent Player A from getting an average win of $V$. For this reason Player A will not compromise with anything less than $V$.

b) There is a strategy that Player B can follow that guarantees that Player B is not going to lose more than the average value of $V$.

c) Since the games are of zero-sum, whatever Player A wins, Player B must lose. Player B tries to limit Player’s A average return of $V$, since Player’s B aim is to minimise his losses.

---

7 Owen Guillermo (1982), 'Game Theory' 2nd ed. p.13-14
8 Owen Guillermo (1982), 'Game Theory' 2nd ed. p.13-14
The last assumption is very important, because in games that are not zero sum type, statement c does not hold. Those games are called non-zero-sum type and the payoff function is not equal in the way that whatever Player A wins, Player B has to lose. It does not hold because Player B is able to limit Player’s A winnings, and Player A will have to do so. This is also the reason that equilibrium points can have different payoff functions in non-zero-sum games and some can be more attractive than others.

More formally, when X and Y are mixed strategies for Player A and Player B respectively, and let $\Gamma$ be the payoff matrix, then:

$$\max_x \min_y X^T \Gamma Y = \min_y \max_x X^T \Gamma Y = V$$

Where $V$ is the value of the game and X and Y are the solutions. When more than one optimal mixed strategy exists, then there exists infinitely many.

Let's take for example the game matrix below:

$$\Gamma = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

The first row is $\begin{pmatrix} 1 & 2 \end{pmatrix}$. The minimum between the two elements is 1. The second row is $\begin{pmatrix} 3 & 4 \end{pmatrix}$. The minimum between the two elements is 3. The maximum between these two minimums is 3. Now let's take the two columns. The first column is $\begin{pmatrix} 1 & 3 \end{pmatrix}^T$. The maximum element between the two is 3. The second column is $\begin{pmatrix} 2 & 4 \end{pmatrix}^T$. The maximum between the two is 4. The minimum between these two maximums is 3. So $\max_j \min_i a_{ij} = 3$ and $\min_j \max_i a_{ij} = 3$, so $\max_j \min_i a_{ij} = \min_j \max_i a_{ij}$. Furthermore, the element $\alpha_{2,1}$, which has the value 3, is the value of this game.

Below is an example of a matrix game that $\max_j \min_i a_{ij} \neq \min_j \max_i a_{ij}$ and that is why equilibrium situations do not exist in this game.

$$\Gamma = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

The maximum of the minimums of the rows here is $-1$. The minimum of the maximums of the columns is 1. That is $-1 \neq 1$.

Even though it is not certain what will happen in this game, one thing is certain: $\max_j \min_i a_{ij} \leq \min_j \max_i a_{ij}$. That is Player A, should win at least the value of his ‘gain-

---

floor’. And also, Player B should lose at the most what the value of his ‘loss-ceiling’ is.

2.2.7 Optimal Strategies

Definition: A row \( i \)th in a matrix \( G \) is said to dominates another row \( k \)th when \( a_{i,j} \geq a_{k,j} \) for all \( j \) and \( a_{i,j} > a_{k,j} \) for one \( j \) at least.

Definition: A column \( j \)th in a matrix \( G \) is said to dominates another column \( m \)th when \( a_{i,j} \leq a_{i,m} \) for all \( i \) and \( a_{i,j} < a_{i,m} \) for one \( i \) at least.

This means pure strategies can dominate other pure strategies, since in a game matrix \( G \), strategies are represented by rows and columns. The undominated strategies in a game are always better than the ones that can be dominated.

So, in a matrix game \( G \) if \( i_1,\ldots,i_k \) rows are dominated, then Player A has the optimal strategy \( X \) with \( X_{i,j} = \cdots = X_{i_k,j} = 0 \). The same holds for Player B and column domination. With all the dominations of rows and columns the matrix game \( G \) becomes smaller and furthermore it is easier for us to work.

Let’s take for example the game matrix:

\[
\begin{bmatrix}
4 & 0 & 2 & 8 \\
2 & 4 & 10 & 6 \\
8 & 2 & 6 & 4
\end{bmatrix}
\]

The third row, \((8 \ 2 \ 6 \ 4)\), dominates the first one, \((4 \ 0 \ 2 \ 8)\).

The 2\textsuperscript{nd} column, \((0 \ 4 \ 2)^T\), dominates the 4\textsuperscript{th} one, \((8 \ 6 \ 4)^T\).

Then the 2\textsuperscript{nd} column dominates the 3\textsuperscript{rd} one.

After these three dominations we get the game matrix \( G \) that is a 2x2 matrix:

\[
\begin{bmatrix}
4 & 0 \\
2 & 4 \\
8 & 2
\end{bmatrix}
\]

\[
\begin{bmatrix}
4 & 0 \\
2 & 4 \\
8 & 2
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 4 \\
8 & 2
\end{bmatrix}
\]

When a saddle point exists, an element \( a_{i,j} \) that is the minimum in its row and maximum in its column, then it is easy to find the optimal strategies. In this case, the pure strategies \( i \) and \( j \) are going to be the optimal strategies for Player A and Player B. Also, if \( X \) and \( Y \) are the mixed strategies of Player A and Player B respectively,

\[10 \] More about domination see: Binmore, Ken. Fun and Games - A Text on Game Theory, 1992-p.146.
and \( X_i = 1 \), and \( Y_j = 1 \), and also the rest of the components are equal to zero, then \( X \) and \( Y \) are going to be optimal strategies as well for Players A and B.

If no saddle points exist, and the game matrix \( \Gamma \) is a 2x2, then we follow the way that solves “2x2 Games”. We get the optimal strategies from the following two equations:

\[
X = \frac{\Gamma^H}{\| \Gamma^H J^T \|}, \quad Y = \frac{\Gamma^H J^T}{\| \Gamma^H J^T \|}
\]

Also we can get the value of the game as well by:

\[
V = \frac{\| \Gamma \|}{\| \Gamma^H J^T \|}
\]

Where \( J \) is the vector \((1, 1)\), \( \Gamma^H \) the adjoint of \( \Gamma \), and \( \| \Gamma \| \) the determinant of \( \Gamma \). The example\(^{11}\) below, it is based on this particular case. Consider the matrix game \( \Gamma \) below with no saddle points and \( \Gamma^H \) being the adjoint of \( \Gamma \):

\[
\Gamma = \begin{pmatrix}
1 & 0 \\
-1 & 2
\end{pmatrix} \quad \Gamma^H = \begin{pmatrix}
2 & 0 \\
1 & 1
\end{pmatrix}
\]

\[
\| \Gamma \| = 2 \\
\| \Gamma^H \| = (3, 1) \\
\Gamma^H J^T = (2, 2) \\
\| \Gamma^H J^T \| = 4.
\]

From the above, we get the following strategies:

\[
X = \begin{pmatrix}
\frac{3}{4} \\
\frac{1}{4}
\end{pmatrix} \quad Y = \begin{pmatrix}
\frac{1}{2} \\
\frac{1}{2}
\end{pmatrix}
\]

The payoff value will then be:

\[
V = \frac{1}{2}
\]

\(^{11}\)The example is taken from Owen Guillermo (1982), 'Game Theory' 2\textsuperscript{nd} ed. p.26
Other type of games are the “$2 \times n$ and $m \times 2$ Games”. In these kinds of Games, one of the two Players has only two strategies. For the $2 \times n$ games Player’s A problem is to make $v(x) = \min_j \{a_{1j}x_1 + a_{2j}x_2\}$ the maximum.

\[
x_1 = 1 - x_2
\]

So:

\[
v'(x) = \min_j \{(a_{2j} - a_{1j})x_2 + a_{1j}\}
\]

Then, since $v(x)$ is the minimum of all the linear functions, $n$, of $x_1$, a single variable, we can plot all these and their minimum $v(x)$ is going to be maximized by graphic methods. Lets take for example the game matrix $\Gamma$:

\[
\Gamma = \begin{pmatrix}
4 & 6 & 2 & 10 \\
8 & 2 & 12 & 0
\end{pmatrix}
\]

The functions $(a_{2j} - a_{1j})x_2 + a_{1j}$ can be plotted easily since they all pass from the points $(1, a_{2j})$ and $(0, a_{1j})$. The plot is going to look like the plot below:

Where the red line represents the $v(x)$ function and X is the highest intersection point of this red line. Since the abscissa of that point equals to $2/7$ and ordinate equals to $17/7$, then we get that the value of this game is $v = 17/7$ and the optimal strategies are $x = (5/7, 2/7)$.

Another type of games is the “Symmetric Games”\textsuperscript{12}. Symmetric games are the ones that their matrices are skew-symmetric. A skew-symmetric matrix is a matrix that its $ji\ ij$ is the minimum of all the linear functions, $n$, of $x_1$, a single variable, we can plot all these and their minimum $v(x)$ is going to be maximized by graphic methods. Lets take for example the game matrix below is skew symmetric:

\[
\Gamma = \begin{pmatrix}
0 & 2 & -1 \\
-2 & 0 & 4 \\
1 & -4 & 0
\end{pmatrix}
\]

Therefore, the above is a symmetric game.

\textsuperscript{12} More about symmetric games can be found in the book: Owen Guillermo (1982), 'Game Theory' 2\textsuperscript{nd} ed. p.28
The value of the game, for symmetric games, is always zero. Furthermore, if Player A has an optimal strategy $X$ then $X$ is an optimal strategy for Player B as well.

Since matrices represent games, elements of matrices can be treated as equations’ coefficients. That is why and how linear programming comes to play.

2.2.8 Linear Programming

Generally, linear programming is a method for handling complex problems arising in the direction and management of large systems. Linear programs are problems of maximizing and minimizing linear functions that are called objective functions. They are focus to linear constraints. The constraints might be equations or variables.

The mathematical form is to find $(x_1, x_2, \ldots, x_m)$ in order to maximize $z$, where $z$ is:

$$z = \sum_{i=1}^{m} c_i x_i$$

Subject to:

$$\sum_{j} a_{ij} x_j \leq b_j$$

with $j = 1, 2, \ldots, n$ and $x_i \geq 0$.

Where $z$ is called the value of the program when maximized. The solution of any matrix game can be represented as a linear program. When Player A uses his strategy $(x_1, x_2, \ldots, x_m)$ then he can expect the minimum a number equal to $\lambda$, where $\lambda$ is:

$$\sum_{j} a_{ij} x_j \geq \lambda$$

For $j = 1, 2, \ldots, n$

So in order to get an optimal strategy for Player A, we maximize $\lambda$ subject to

$$-\sum_{j=1}^{n} a_{ij} x_j + \lambda \leq 0$$

with $j = 1, 2, \ldots, n$ and

$$\sum_{i=1}^{m} x_i = 1$$

with $x_i \geq 0$ and $i = 1, 2, \ldots, m$. 

21
2.2.9 Simplex method

Many activities of organisations can be seen and represented as linear programming problems as described above. Simplex method is a way that George Dantzig represented in a conference in 1949 for solving these problems, where it became clearer that the application where this was to be applied was quite wide. “The simplex method is an iterative procedure that proceeds from a vertex of the feasible region to an adjacent vertex, moving in a direction that improves the objective function”\textsuperscript{13}

Over the last 40 years, they have only been some computational improvements have been made. Recently some methods called interior point methods have been developed, which are effective in some problems.

The detailed algorithm is difficult to be described precisely here, because of the various choices of basic variables and rows that the notation copes in the tableau that they will be solved.

But for a better understanding a geometrical example\textsuperscript{14} will be used here. For example to maximize \( w = 2x + y \) subject to:

\[
\begin{align*}
x & \leq 1, \\
y & \leq 1, \\
2x + 2y & \leq 3, \\
x, y & \geq 0.
\end{align*}
\]

Consider the following hyper polyhedron below.

The area shaded in grey is the constraint set. The slope of the objective functions is shown by the arrow. Starting from the origin, \( w \) is getting increased along any of the edges that start from the origin. If we take for example the edge that ends at \( (1,0) \), then \( w \) is increased towards \( (1,1/2) \). Also, at this point, \( w \) decreases along both of the edges that meet at \( (1,1/2) \). Therefore, the point \( (1,1/2) \) is the solution. The simplex method solves this kind of problems algebraically.

\textsuperscript{14} The example is taken from Owen Guillermo (1982), ‘Game Theory’ 2\textsuperscript{nd} ed. p.41-42
The simplex algorithm can be found in detail with many examples as well as the simplex tableau and the simplex method in Michael Brigden's lecture notes for the MA30087-Optimization methods of Operational Research module at university of Bath.

2.3 Summary

People over the years solve game theory problems in many different ways depending on what the type of the game is. It was concluded that the best way of solving any given game matrix $G$ that is of two-person zero-sum type, is with the use of linear programming and with the help of the simplex method. Examples of calculators that solve game matrices can be found on the Internet at:

URL: ‘www.mkaz.com/math/MatrixCalculator.java’
In this chapter, all the methods that were used for collecting and gathering the requirements needed for the development of the system are explained from the initial steps to the complete and final requirements specification…
3 Requirements Analysis & Requirements Specification

3.1 Requirements Analysis

3.1.1 Introduction

This section is concerned with an in-depth analysis of the project, system domain. Requirements by definition is what the system is suppose to do, in what way it will do the various tasks that the users require, as well as things the system should not do. It is the functionality of the system. It is of great importance that the developer will truly understand what the stakeholders want from the system from the initial stages in order to develop the system correctly fulfilling the users desires. The failure of many systems has been traced back to the requirements elicitation and analysis. So it is of vital importance not to just get the requirements right, but get the right requirements as well.

3.1.2 Scope of the product

The scope is to develop a program that will be able to calculate the value of any given dimensions matrix game. The product will also be able to find the optimal strategies for the two players as well as give to the user feedback for every step of the procedure made in finding the final solution. This product can be used by maths students, computer science students, who want to check their results, by lecturers to teach game theory or by anyone that is interested in game theory. Also, this system can be used from economists to find equilibrium points for some strategies that they may want to pursue.

3.1.3 Purpose of the product

The purpose of this product is to give feedback and quick solutions to matrix problems, value of a game, as well as optimal strategies for the two players. The product can be used for educational purposes as well as for a quick calculator for game theory problems. Also, if the data will be given in another format the calculator can solve linear programming problems with simplex method.

3.1.4 Requirements Elicitation and Analysis

Requirements elicitation is the process that the requirements can get discovered and gathered in order to guide the developers in developing the system right and also most importantly the right system.
3.1.4.1 Sources of requirements

The first method that was used was interviews with potential users, such as computer scientists, mathematicians, PhD students, economists and lecturers of the domain. Notes were kept in each interview for later use and analysis. By doing that though it was realised that something more was needed, since these people were the actual people that would use the system in the future.

From the interview notes were collected, some guidelines were raised. Then with the help of the notes and some imagination of the system some questions were written in order to get even more useful information from the users. The second method that was then used was questionnaires. Questionnaires were given to people with different background in connection with game theory and linear algebra. People that do or did maths in the past were more familiar so they were preferred. That is why more than 50 percent of the questionnaires were given to them. Some of the questionnaires can be found in the appendices.

After collecting back all the questionnaires and reading through each one of them what all the people said and wrote, a very good picture of the system started getting created. After the first two methods, a third method was used to get a better approximation for what had to be developed. This did not involve users. The method used then was the method of brainstorming. Having all the information inside a room, a blackboard was used to write on it different ideas for the creation of the system. Staring the blackboard for a few hours and by reading again and again questionnaires and notes a complete picture of the system requirements was developed.

Additional to the above three methods, a supplementary method was used as well. That was the evaluation of already existing systems. It was really helpful for finding out what other people did in the past and how they approached different aspects of the problem as well as different methods that they have used for solving the problem.

---

15 Ian Sommerville, Software Engineering, 6th edition, p.125
Although it did not have the users involvement like the first two methods. Some of them were really efficient in contrast with some others that were really slow and inefficient. Some of them were giving wrong values as well. Examples of some of the systems that were evaluated can be found in the appendices.

The four methods used were proven to be very efficient since in a relatively quick amount of time requirements were gathered and a picture of how the system would work was generated. Moreover, in the questionnaires a question was included for what the system should not do. In this way information was gathered for what to stay away from and how to stay away from it, in the designing process.

After the elicitation of the requirements two main methods were used for analysing these requirements. The methods used were the ‘CRC’ cards and the Use Cases method.

**Definition:** “A Use Case is a sequence of transactions in a system, whose task is to yield a measurable value to an individual actor of the system.”

Use cases are really good in representing the functionality of the system throughout. Each use case that has been used for this project includes a title, a primary and a secondary actor, precondition, goal in context, scope and level, triggers, success end condition, failed end condition and priority. The use cases can be found in the appendices.

**Definition:** “A Class Responsibility Collaborator (CRC) card is a physical index card that is annotated and used in a group setting to represent a class of objects the behaviour and interactions of both class and object.”

### 3.2 Requirements Specification

In this section, all the requirements that were gathered from the interviews with potential users, questionnaires, evaluation of existing systems and brainstorming are included, as well as requirements that arise from the literature review.

The product is going to consist of seven main parts. The first part is going to be the “mode selection” that will give the user the option to select whether feedback is to be provided or just quick solutions and results. Then the “matrix dimensions insertion” will follow, which here the user is going to specify what dimensions the game matrix will have. After that a “confirmation step” will follow, where the system is going to display to the user what dimensions have been inserted and ask the user whether to proceed or not. Then the “matrix elements insertion” will follow where the user puts in the system all the elements of the game matrix and again a section follows for “confirmation” where a complete display of the game matrix is shown to the user. Then at the next step, the system “checks” how the specific problem can be solved.

---

16 Jacobson's Ivar Object-Oriented Software Engineering: A use case Approach (ACM Press S.)
17 Brown, Pad Lecture notes for K268 SENG2100: Object Oriented Analysis and design. Dublin Institute of Technology, 2004
and informs the user about the solution method that is going to be performed. Then
the system will solve the problem in the way that is appropriate and as for the “final
step” the system will give back to the user the value of the game together with the
optimal strategies for the two players and any supplementary feedback needed for the
user to understand how and why the solutions where generated.

3.2.1 Requirements drawn from Literature Review

The requirements that are described in this section, are the requirements that come
from the literature review. They are more concerned with the mathematical theory
behind the program.

1. The program must be able to solve any dimension two-person zero-sum
game.

2. The program should be able to decide if any saddle points exist within a
matrix game.

3. The program should be able to find the saddle points when they exist.

4. The program should be able to use the simplex method in the case of
absence of saddle points.

5. The program should be able to maximize a linear function subject to linear
constraints.

6. The program should be able to minimize a linear function subject to linear
constraints.

7. The program should be able to find the value of the game by maximizing
(minimizing) a linear function, using the simplex method.

8. The program should be able to find the optimal strategies for the two
players using the simplex method.

3.2.2 Functional Requirements

Functional requirements are the requirements that describe the functionality that the
system provides.

1. The user must be able to put any matrix dimension, m×n, that the user desires.

2. The system must include a method for error checking on the matrix
dimension. This must include:

   a. The system must understand when the user did not put an integer or a
      positive number for the row dimension of the matrix. Also the system
must understand when a zero dimension was given as input for the rows of the matrix.

b. The system must understand when the user did not put an integer or a positive number for the column dimension of the matrix. Also the system must understand when a zero dimension was given as input for the columns of the matrix.

c. The system must understand when the user did not put numbers for the row and column dimensions of the matrix.

d. The system should also be able to check that the user did not leave any of the fields blank.

e. System must be able to explain to the user what the error was in the case of the existence of an error.

3. The system should provide the user a facility for selecting between different modes.
   a. The system should provide a mode for providing feedback.
      i. The system should be able to show all the tableaus and pivots chosen for each tableau to the user together with the pivoting step at each step of the algorithm.
   b. The system should provide a mode for quick results.
   c. The system must understand and not fail when the user has selected an invalid option for the mode or when the user left the selection field blank.

4. The user should be able to enter any number when inserting the elements for the game matrix.
   a. The system must include a method for checking that the user has entered numbers and not something irrelevant.
   b. The system should be able to accept any double numbers for the elements of the matrix.
   c. The system must be able to accept negative numbers for the elements of the matrix.
   d. The system must be able to check that the user did not leave any of the fields blank.
   e. System should be able to explain to the user what the error was in the case of the existence of an error.

5. The system must display to the user the game matrix, dimensions and elements, which has been inserted before solving the game.
6. The system must be able to check how the problem can be solved. That is either by finding a saddle point, which is when minmax is equal with maxmin, or if not then by using linear algebra and the simplex method.
   a. The system must display to the user how the game matrix is going to be solved.
   b. The system must be able to proceed to the method that is going to be used for solving the problem automatically.

7. The program must be able to solve the problem by any method needed for solving the desired game.
   a. The program should be able to give feedback for what is going on to the user.

8. The system must be able to display to the user the final result, the value of the game, together with the optimal strategies for the two players.

9. The system must provide a way for the user to exit the program at any given time.

10. The system should work in a way that minimises the possibilities for errors from the user. System must interact with the user only when necessary.

11. The system must guide the user in a straightforward manner.

3.2.3 Non-Functional Requirements

Non-functional requirements are the requirements that are not concerned with the functionality of the system. They do not directly consider the specific functions that are going to be delivered by the system. They are separated into three main categories:

- Product Requirements
  - These specify the behaviour of the product.

- Organisational Requirements
  - These come from organisational policies and organisational procedures.

- External Requirements
  - These come from irrelevant factors. They are external to the system and the system's development process.

3.2.3.1 Product Requirements

- Usability Requirements
The system will be developed according to the six usability principles specified by Ian Sommerville. These are:

- **User familiarity**
  - The interface should use terms and concepts that come from the experience of the people that will make the most use of the system. In the two-person zero-sum calculator case, the people that are going to use it more frequently are mathematicians, economists and computer scientists. So the system must have a consistent UI throughout the program.

- **Consistency**
  - The interface should be consistent. Same operations should be performed in the same way. For example, the user will have to put the matrix dimensions. Later when the system will require the user to fill the matrix, the system should use the same way for asking the user to do it.

- **Minimal Surprise**
  - Users should never be surprised by the behaviour of the system. For example if the user inserts zero for the dimensions, system should explain that this is not allowed, and not throw the user out of the program.

- **Recoverability**
  - The interface should include mechanisms to allow the user to recover from errors. For example a “Cancel” button can be included in the system for the user to cancel an operation.

- **User Guidance**
  - The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities. In the case of the calculator a detailed step-by-step description of the tableaus together with all relevant information for what and why each thing the program did.

- **User Diversity**
  - The interface should provide appropriate facilities for different types of the system users. The system should therefore provide facilities for novice and experienced users of the system, without hindering either one or the other. For that purpose the system will have two modes in the beginning.

- Also apart from those requirements, some other usability requirements that are not specified in the above are:
- Going from the starting point to the end point should be easy for any kind of user and the system should be able not to give many and complicated options to the user but to do everything for him/her. That is why the calculator will ask the user only when something is needed at the time needed.

- The system should also give the user options that are easy to choose, such as “OK” in order to proceed with the operation, or ”Cancel”. In that way mistakes are minimised and user does not have to think of what to do.

---

### Efficiency Requirements

- **Performance requirements**
  - The system must be quick and correct
  - The system must be able to find solutions in less than 2 seconds from the time the user clicks enter.
  - The user should be able to restart from the beginning in less than 3 seconds.

- **Space Requirements**
  - The program should take no more than 5 Mbs to store on the computer.

---

### Reliability Requirements

- The system will not fail when completely wrong data has been inputted.
- The system will automatically proceed to the next stage if possible, otherwise it will tell the user what went wrong and how that can be avoided next time.

---

### Portability requirements

- The system should be able to run on any operating system including Windows, Linux, and Mac operating environments.
- The system should be able to run even in slow computers with 32 Mb of RAM.

---

### 3.2.3.2 Organisational Requirements

- **Delivery Requirements**
  - Deliver Requirements
The system must be operating fully by Monday the 16th May 2005.

The deliverable shall consist of a single document in seven parts: the literature review, requirements document, design document, detailed design and implementation document, test plan with testing and a user manual.

A securely attached and clearly labelled CD-ROM containing the calculator ready for installation.

- **Implementation Requirements**
  
  - The programming language that is going to be used is Java.
  - The platform, environment that is going to be used is Eclipse

3.2.3.3 **External Requirements**

- **Ethical Requirements**
  
  - The system should not be offensive within any culture where the system will be made available.

3.2.4 **Hardware Requirements**

Hardware requirements are the requirements that are describing what the system needs in order to operate in connection with the hardware.

- The system will require:
  
  - A computer with:
    
    - Any processor
    - Any operating system (e.g. Windows, Linux, Mac)
    - 32 Mb of RAM
    - 20 Mb of hard disk available
    - A keyboard
    - A mouse
    - A monitor
    - Optionally a printer so that all the results of the matrix game can be printed together with the feedback for each game.

3.2.5 **User Requirements**

- Some modest knowledge on linear algebra
- Some knowledge on game theory
- Some basic computer skills
- Interest in the area
3.3 Requirements Validation

Requirements Validation method, or activity, is for checking that all the requirements are realistic, consistent and complete. Therefore, performing this activity, errors are being discovered in the requirements. These errors have to be modified and corrected. Additional to that even new requirements will come to light during the performance of this activity.

Requirements validation is extremely important because if there are errors in the requirements document, this can lead to extensive rework especially when the errors are discovered later during development or even worst during testing.

The tests\textsuperscript{18} that are involved with the requirements validation include various checks like:

1. Validity checks
2. Consistency checks
3. Completeness Checks
4. Realism Checks
5. Verifiability Checks

To be able to deal with all these checks, Somervelle suggests a number of techniques that can be used either in conjunction or individually to ensure that all the checks are made in the correct and appropriate way. These techniques are:

1. Requirements reviews
   - Requirements are analysed systematically by a team of reviewers
2. Prototyping
   - By demonstrating to the users a model of the system and then users can comment on that model whether it satisfies their needs or not.
3. Test-case generation
   - Since requirements are ideally testable, if a test is difficult to be designed for, then it must be difficult or impossible to implement it. So that requirement will need reconsideration.
4. Automated consistency Analysis
   - When the requirements are expressed as a system model, then CASE tools can be used to check if that model is consistent. The figure 3.3.1\textsuperscript{19} below shows an automated consistency checking of requirements.

\textsuperscript{18} Ian Sommerville, Software Engineering, 6\textsuperscript{th} edition, p.137
\textsuperscript{19} Ian Sommerville, Software Engineering, 6\textsuperscript{th} edition, p.138
All these four methods are used in conjunction with each other for the purpose of the calculator requirements. The requirements evolution in connection with time can be seen in the figure 3.3.2\textsuperscript{20} below.

\textsuperscript{20} Ian Sommerville, Software Engineering, 6\textsuperscript{th} edition, p.141
In this chapter, it is explained how the system is going to be developed. That involves how the system is split into units and how these units are going to be developed as well as how these units work, since some units involve methods and algorithms...
4 Design

4.1 Introduction

There are many ways of performing a software process and many models for all the different software process approaches. The model that was used for the purpose of the calculator development is the waterfall model. This model takes all the core process activities of specification, development, validation and evolution and represents them separately as requirements specification, software design, implementation and testing as separate process phases.

This process is widely used for practical systems development. However, “this approach is likely to be influential in the 21st century as assembling systems from reusable components is essential for rapid software development.”\(^{21}\) For the purpose of the calculator development it is the most suitable one. The waterfall model, which takes its name from the cascade phase-to-phase structure, is also known as software life cycle.

Figure 4.1 The Waterfall Model\(^ {22}\)

4.2 Module Overview

For better understanding, the system has split up in different modules that together construct a complete system. Each module can be interpreted as a function or a collection of functions, that together structure the functionality of the system. Each module in this system depends upon another, but each one of the modules has a unique role and can be implemented and tested individually. The dependencies of the

---

\(^{21}\) Ian Sommerville, ‘Software Engineering, 6th edition’, p.44

\(^{22}\) Ian Sommerville, ‘Software Engineering, 6th edition’, p.45
system as well as the general and complete system structure can be seen in the figure below:

**Figure 4.2 Modules Overview and modules relationships:**

Mode module – This is the mode that the user is going to select for the program to follow. It will have two options available to the user and these options will be either to select a feedback mode with explanations of every step the program follows in order to find the solutions or either a quick mode that will only provide the user with quick solutions.

Data module – This module is included basically for handling the users input. It will contain routines for truth checking and also routines that will display to the user what he/she has inputted.

MaxMin module – This is a utility module that checks if the game matrix that the user has inputted contains an element, which is itself a saddle point. That means maxmin is equal with minmax. If such an element exists then it directs the flow of control of the system to the solutions module, the last module of the system, since the solutions can be read off.

Simplex method module – This module is the simplex method algorithm module. The flow of control reaches this module when saddle points do not exist and the program has to work another way for finding the solutions. It is itself a function of functions since the simplex method is a nested operations and steps algorithm.
Solutions module – This module is the final module of the system. The flow of control ends there independently of the direction that was followed. This module is responsible for reading off the solutions. Then it displays the solutions to the user.

4.3 Data Structures

For the purpose of this program different data structures are going to be used. The main one is a hierarchical structure that will be used for placing and positioning the elements of the game matrix. Also, that hierarchical structure is going to be used for storing numbers in the tableau’s that are going to be used when performing the simplex method. Simple small data structure are going to used throughout the program for the storing of things like mode selection and labels of the final solutions.

The hierarchical structure was chosen for the game matrix because it works like a tree. Since any given matrix is of 2 dimensions, then for every element in a row of that matrix, a series of more elements correspond to that element. For example the matrix below is a 4x3 matrix. The first row is composed from three elements. These are the elements “1”, “2”, “3”. For every one of these elements, another three elements correspond to them. The tree representation for each node looks like the one below:

Figure 4.3.1

Therefore, for the game matrix that the user is going to insert, the first row is going to be stored and then for every element of that row, a column of elements is going to be stored that corresponds to each element.

The same hierarchical structure is going to be used for storing the tableaus of the simplex method. This can be seen in the Figure 4.3.2 below.

That is, because every object in the first row will have a connection with every object in its column. Furthermore, when a comparison will be needed among all the elements of the same column for finding the pivot, this way is going to be more efficient, faster and easier to implement.
A small structure will be used for other sections of the system. The mode system for example will use a simple binary structure. After the user has selected the option that is desired, the system will store that as a simple Boolean operation and, as the system proceeds, in every step the program will recall the mode and act in the appropriate way.

A small structure will also be used for ordering and storing the labels of each one of the optimal strategies of the two players. The figure below describes that:

Although the ordering and storing of the labels uses a small structure, later when the results will have to be stored and be displayed to the user in an ascending order, the system will use a hierarchical structure of the form that is displayed below:
That is for every label of the game matrix, a value will correspond to it. This can also be seen as a tree with a one-child parent relationship. The diagram for this can been in figure 4.3.4

4.4 Mode Selection

This will provide the user with two main selections. First selection is to proceed with feedback. This option is going to provide the user with a step-by-step description as well as general useful information for checking, learning and understanding the basis of game theory and linear programming and the simplex method. The second option available to the user is going to be a quick mode selection. By selecting this option, the user will be able to get quick solutions without getting feedback steps and generally information that will delay him from getting the solutions straight away.

The mode selection is going to be implemented by holding into the memory of the system the selection that the user made. As the system will proceed, the system will remember the selection made by the user and will act the appropriate way.

For example when the quick mode is selected, the system skips all steps that are involved with any output that was going to be displayed to the user as well as other feedback information and performs only those steps that are concerned in finding the solutions. In the same way, if the feedback mode is selected, the system will identify that, and in every step that will be performed the system will give the appropriate feedback to the user.

4.5 Minmax Maxmin Method

This important method will take the game matrix that the user will give to the system and will perform a check. It will check if maxmin is equal to minmax. If the method finds that the two are equal, then that means that the particular element of the game matrix is a saddle point and that also means that the program will not need to proceed with any further procedures in order to find the result.

For example in the 3x3 matrix below, the maximum from all the minimums is seven. The minimum from all the maximums is seven as well. So both maxmin and minmax are 7, therefore equal. So since a saddle point exists, the element $A_{31}$ of this matrix,
then there is no need for preceding any further. The program will skip everything that is followed and go to the final section where the solutions are.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
7 & 8 & 9
\end{bmatrix}
\quad
\begin{bmatrix}
1 \\
4 \\
7
\end{bmatrix}
\]

If saddle points do not exist, then the function will understand it and it will pass the flow of control to the simplex method, a method that includes an algorithm for finding the solutions. That is the value of the game and the optimal strategies for the two players. Below pseudo code is provided for the maxmin function.

### 4.5.1 Pseudo-Code for Maxmin function

```plaintext
temp = A_{11} ;
temp2 = A_{M1} ;

for ( A_{11} to A_{IN} )
{
   for ( A_{ij} to A_{ij} )
   {
      if( temp > A_{ij} )
      temp = A_{ij}
   
      if ( temp2 <= temp )
      { temp2 = temp; }
   }
}
```

In a similar way, the system will include a function that is called Minmax. This function, will find the maximum elements from all columns of the game matrix and then the minimum among them, the opposite that Maxmin did, which is finding all the minimums from all the rows and then the maximum among them.

After both functions find solutions, an operation is going to be performed to check whether these two functions give back the same number. If the number is the same then system will understand that saddle points were found. The system will then pass the flow of control to the final step of the program, to give the user the solutions. If the number is not the same, then it will pass the flow of control to the simplex function as stated above in order to work out the solutions.
4.6 Simplex Method

Simplex method is the most important method of the system. It is basically a method that will perform the simplex algorithm. The flow of control reaches this method when no saddle points exist in the game matrix that the user has inserted. Then, this function takes the game matrix that the user has inserted and from the elements of that matrix the system creates a semantic tableau. Lets take the example of the 3x4 matrix:

\[
\begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{bmatrix}
\]

The system will then take this matrix and it will create the following semantic tableau, which will also be the starting point of the simplex method:

**Figure 4.6**

\[
\begin{array}{cccc}
X1 & X2 & X3 & X4 \\
\mathrm{Y1} & 1 & 2 & 3 & 4 & 1 \\
\mathrm{Y2} & 5 & 6 & 7 & 8 & 1 \\
\mathrm{Y3} & 9 & 10 & 11 & 12 & 1 \\
-1 & -1 & -1 & -1 & 0
\end{array}
\]

If negative elements exist in the original game matrix the system will make them positive by adding a number which is the same as the biggest negative. In that way the elements numbers in the matrix will be from zero and more. That number will be subtracted at the end, after finding the final tableau.

The system will then follow a procedure for choosing a pivot. As long as negative numbers exist in the last row, the system will choose any column. After choosing column the system will divide each element of that column with the respective row element of the last column. The element that will have as a result the smallest positive among the three will be selected to be the pivot. This operation is going to be recursively done in the system until no negative elements remain in the last column.

After finding the pivot, the system will then perform an operation that is called the pivot step. Basically the system will change all the elements in the matrix by doing the following calculations:

- The elements that belong to the pivot’s row will be replaced with an element that is the result of the original element divided by the pivot.

- The elements that belong to the pivot’s column will be replaced with an element that is the result of the original element divided by the pivot, together with a sign change.

- The elements that do not belong to pivot’s row or column will be changed with new elements that are the result of the original element minus the product
of the elements that are in the same row and column positions of the pivot, divided by the pivot.
That is: \(A(i,j) = A(i,j) - A(p,j)A(i,q) / A(p,q)\)

- The pivot element it is going to be replaced by its reciprocal.

Symbolically the above operations would be:

\[
\begin{pmatrix}
a & b \\
c & d
\end{pmatrix} \rightarrow \begin{pmatrix}
1/a & b/a \\
-c/a & d - (bc/a)
\end{pmatrix}
\]

Where \(a\) is the pivot, \(b\) is any number in the same row as the pivot, \(c\) is any number in the same column as the pivot and \(d\) is any number that does not belong in the pivot’s row or column.

After the system has finished with the pivot step, it will exchange the X and Y labels that exist on the left column and at the top row of the pivot, respectively, as shown below:

The system will continue to recursively perform the three last operations, selection of the pivot, the pivot step and the label exchange steps, until all the negative elements in the last row of the tableau become positive.

After performing the recursive operations, at some point the system will detect that no negative entry remains at the bottom row. At that point, the system will manage to get the final tableau. The final tableau contains the optimal strategies for the two players as well as the value of the game matrix.

The value of the game can be found at the bottom right-most place of that tableau. It is the reciprocal of that element. If the system has already added a positive big number in the beginning it will subtract it here.

The optimal strategies for the two players can be calculated by dividing the values of the elements of the bottom row with the value of the game, for the first player. Similarly for the second player, we divide the elements of the right column with the value of the game. In this way the system will find the probabilistic answers for the two players.
4.7 User Interface Design

The user interface design is very important, since this is a calculator for people. Users must find it simple and easy to use. It is very important that the system will be designed with an interface that is straightforward to use, that it will literally guide the users through all the steps of the system in finding the solutions. This section is divided into two sub-sections. The user input section and the user feedback section.

4.7.1 User input

The system’s usability is based on the input that the user is going to provide. The way that the system is going to interact with users is through the mouse and the keyboard. Different operations will be performed in different ways. For example, the matrix game that the user will give to the system will be given by typing numbers from the keyboard. When the user will finish typing he/she can also use the mouse for pressing an “OK” button for the system to accept the matrix and proceed to the next step. Also, the user will be able to press the “enter” button for complete keyboard interaction instead of using the mouse.

4.7.2 User feedback

The system will give the user feedback for several different things. Two main methods are going to be used. The first method is with the help of a console, which is going to keep the user up-to-date with what is going on behind the scenes. Another method will be the use of pop ups for explaining and guiding the user throughout the program.

4.8 Error Handling

As the error handling is important, the system will have mechanisms for recovering and retrieving from possible errors. For example, when a number format is not appropriate, the system will explain that to the user and exit, rather than showing to the user a big series of numbers and say that was the error.

The system will also contain mechanisms to retrieve from bad user input. Whatever input the user puts even if that is completely irrelevant of what he/she should have inserted the system will understand it and exit. The system will give the user feedback for what went wrong in a nice and polite way.
In the following chapter, all methods routines functions and algorithms that were used for the system’s development are described and explained in detail. Pseudo code is provided within the chapter for a better understanding…
5 Detailed Design and Implementation

In this section, the procedures for implementation as well as decisions that were taken for producing the two-person zero-sum game calculator based on the design chapter that was described previously are discussed in detail. First, the tools and platforms are described and then the what follows is the important and more difficult steps that were more harder to implement and more vital in the system’s development procedure.

5.1 High Level Implementation Decisions

After a short investigation of what was available, the high level decisions were made for what and how to be used. The use of a top down method was also decided on because it seemed to be most appropriate since the system was divided into modules and these modules could be tested individually later on in the implementation and in testing.

The implementation order, as stated in the design section, of the components is as follows:

- Mode component
- Data component
- Maximin Minimax component
- Simplex method component
- Solutions component

Testing was carried out through all the stages of the top down development method when every one of the components above was implemented following the exact order that is stated above.

5.1.1 Platform

A decision was made about the system to be targeted in a Windows platform. That was mainly decided because the system would be mostly used by mathematicians and all the mathematicians that were involved at the requirements procedure stated that they do not like the Linux environment. Since the author is a computer scientist and familiar with the Linux environment it was decided that the system be compatible with both environments so the procedures that would be followed for the development would make the system compatible with all main platforms. This platform choice had a great influence in the other tools that would be used for development and implementation. These are described in the sections followed.
5.1.2 Implementation Language

After considering the availability of languages, Java was decided to be the language for implementation. Java is a language that can directly interface with all the main platforms. Additional to that, the Java language offers an in-depth and various array manipulation methods that were extremely useful for the purpose of the development of the game theory calculator since matrices can be stored and manipulated in a Java’s 2 Dimensional arrays. Furthermore, the author has some familiarity with Java and array manipulation and that made implementation faster. The alternative language was C due to the fact that the C language has pointers that may have been helpful when manipulating the tableaus and selecting the pivot, as well as when the pivoting step operation would have been implemented. The overall advantages though made Java to be selected as the one for the purpose of the implementation of the system. The decision about choosing Java influenced the selection of other tools as well.

5.1.3 Development Environment

The integrated development environment (IDE) that was chosen for developing and implementing the system is the Eclipse 3.0.1 platform with the Java plug-in. Eclipse has an excellent source browsing facilities that together with the syntax highlighting options that are provided make Eclipse very usable and user friendly. This made the development of the system quicker since the production of the code was done at a quicker rate. Furthermore, the use of this environment did not limit the requirement of operating system compatibility and the system did not loose its independence.

5.2 Overall System architecture

With the help of the requirements specification and the design section, the overall architecture of the system was decided. The overall structure is described in the next paragraph.

After the user runs the program, the system displays to the user a popup page, which is a welcome page explaining what the program is all about and also in this popup page the user is asked what behaviour the system should have according to the user’s preferences. In this popup page there is a blank field that the user can use for typing what the option the system must follow. Two buttons are provided, one for accepting the user’s selection and proceeding and one for cancellation. After selecting the behaviour that the system will have, the system displays to the user a confirmation of the behaviour that was selected in the Eclipse console and the system continues by asking the user to define the dimensions of the game matrix that the system will solve. Afterwards, the user enters the rows and the columns dimensions in the appropriate fields that the system provides. Then the system gets these numbers and performs a truth check on them, so as to see if the numbers are valid. For example, a dimension with the zero number of rows is not a realistic dimension, as well as a negative number for defining the dimensions would not be appropriate. When the system approves the numbers that the user has inserted for the game matrix dimensions, a confirmation page is displayed to the user showing the dimensions entered, together
with a button for proceeding and a button for cancellation. When the user proceeds, the system guides the user to fill all the elements of the game matrix. In the fields that are provided to the user for entering the elements of the game matrix, the user is able to insert any number, negative or positive, with decimal or not, even zero for some elements of the game. This is because the system must accept any number for any element of the game. The system at this point will also check that the user has not inputted a string in the place of a number. After the completion of the construction of the game matrix, the system then displays to the user the game matrix that was entered for confirmation, and again as for the dimensions of the matrix, this screen has a button for proceeding and a button for cancellation. After the system has received the ok to proceed, it gets that matrix and with the minimax-maximin method function, it searches the game matrix for saddle points. If this method succeeds and saddle points exist then the value of the game is displayed to the user together with the solutions. If the method fails in finding saddle points the system continues by passing the flow of control to the simplex method function. The system at this point informs the user with a popup screen that saddle points do not exist and that simplex method is going to be used for solving this game matrix. The system then finds the solutions with this method and it then displays to the user the optimal strategies for the two players together with the particular value of the game. At all stages a console is used for a step-by-step description of what the system is doing and how the system does it.

This is the overall structure of the system. A more detailed description for all the important components, methods, algorithms and functions is going to be described in what follows.

5.3 Mode method

The first page of the system is the mode page. This popup page appears after the program has started, consists of information about the system, a mode selection, a blank text field, two normal buttons, and a small exit button that is located at the top right corner. The information about the system mainly informs the user that this system’s usage is for solving two person zero sum games. Below that in the page, it displays that the user has mainly two options available for what behaviour the system can follow. These two options are the feedback mode and the quick mode. These two options are built in a Yes No manner, a binary way. That is, the system asks the user to fill the text field with a “Y” letter for “Yes” to give feedback. Thus the system follows a feedback mode, or an “N” for “No” to give feedback meaning a quick mode for quick results and solutions only. When the user has presses the “OK” button for the selection, the system stores that selection in a string variable and as the system proceeds, in every main step, if in that string the value is “Yes” then feedback is provided in the Eclipse console. If the value of the mode string is a “No” then all the operations are performed normally as before, apart from the ones that involve printing out feedback to the user.

Below in table 5.3 some pseudo code is provided showing how the mode function is implemented.
Table 5.3

```java
String Mode= JOptionPane.showInputDialog("Press Y for feedback");
String Y="Y"
...
...
IF ( Mode.equalsIgnoreCase(Y))
{
    System.out.println("Feedback mode selected");
}
...
...
IF ( Mode.equalsIgnoreCase(Y) )
{
    System.out.println("This here is some feedback");
}
```

Table 5.3 shows the case that the user has selected the “Y” for yes to feedback. If the user’s selection is “No” then the bottom line “This here is some feedback”, is never be displayed. Similarly, this method appears throughout in the program in all the places that feedback can be provided. The system also has some methods and routines for testing whether the user inserts invalid options or not. How the system is implemented for doing that, will be explained and described in the section 5.5, truth checking methods, that follows.

### 5.4 Data Input methods

After the user has chosen a mode for the system behaviour, the system guides the user for providing the data that is needed for the game matrix to be solved. First the system asks for the game matrix dimensions and then for the actual elements of the matrix.

#### 5.4.1 Game Matrix Dimensions

In order for the system to get the matrix dimensions from the user, the system asks the user to insert the number of rows and the number of columns to blank text fields that the system provides with some option popup panes. The numbers that the user gives for the matrix dimensions, that is for the rows and the columns of the matrix, are stored in two string variables called rows and columns respectively. Then the system parses these two numbers into integers, since dimensions have to be in an integer form. The system, at this point performs check operations to be sure before continuing that the user has inserted valid numbers for the dimensions. How the system performs these validity checks, will be described in the section 5.5, truth-checking methods, that follows. Pseudo code for how this is done is showed in table 5.4.1.1.
Table 5.4.1.1

String rows = JOptionPane.showInputDialog("Enter the number of rows: ");

Similarly, the system gets the number of columns from the user as for the rows. After
the insertion the system parses the two numbers in a way that the table 5.4.1.2 shows.

Table 5.4.1.2

int Matrix_Rows = Integer.parseInt(rows);

After the system does that for both rows and columns dimensions it stores these
numbers in two integer variables and uses them later on, in the construction of the
game matrix.

5.4.2 Game Matrix Elements

After the system checks and confirms that the dimensions that the user has given are
valid it proceeds in getting the full game matrix from the user. First, the system
constructs an array for storing the elements of the game matrix. This array is a two
dimensional array. The size of this array gets defined from the variables that the
system holds for rows and columns from before. Furthermore, this array accepts not
only integers but double numbers as well, negative numbers, zero numbers, and
generally all type of numbers that a game matrix can have.

Then the system displays a popup screen to the user asking him to insert the first
element of the game matrix. In this little popup screen two buttons are provided to the
user, one for the system to accept the first element, and one for cancellation. After the
user enters the first element, the system continues asking the user in a recursive way
to fill all the elements of the game matrix. At this point the system, with the help of
an exceptional handler, makes sure that the user has not inputted something that is not
of a numerical type (i.e. a string). When this is the case, the system identifies this and
informs the user for the invalid insertion. When the array of elements is filled
completely, that means that the game matrix is ready, and the system informs the user
that the game matrix is ready.

The table 5.4.2 below shows how this is implemented with the help of some pseudo
code.
Table 5.4.2

```java
... double[][] matrix = new double[rows][columns]; ...
... For (int i=0; i<matrix.length; i++)
{ For(int j=0; j<matrix[i].length; j++)
{ String Aij = JOptionPane.showMessageDialog("Enter Aij");
 double Element_Aij = Double.parseDouble(Aij);
 matrix[i][j] = Aij;
}
 ...
```

The pseudo code that is provided in the table above is not exactly how this was implemented in the code when the system was created due to the fact that the arrays in Java start from zero. That means some extra variables were used for counting and therefore placing the right element of the game matrix in the right position of the 2D array.

5.5 Truth checking methods

In this section, the various methods for truth method are described. The methods for truth checking were used throughout the system for different parts and different operations.

First, in the mode selection function in the initial page of the system, the program checks whether the user has inserted something or not. Then it checks if what the user has inserted is a valid selection or not. The pseudo code in table 5.5.1 describes how the system manages to perform these operations.

Table 5.5.1

```java
If ( (Mode==null) || (Mode.length()==0) )
{ ... }
Else If( Mode.equalsIgnoreCase(Y))
{ ... }
Else If(Mode.equalsIgnoreCase(N))
{ ... }
Else
{ ... }
```
When the system is sure of what the user has inserted then it stores it and continue. If the system understands that the user did not behave the way he/she is supposed to concerning the input, then the system tries to either recover and proceed, or if necessary exit. But the system in no circumstance stops working.

Concerning the game matrix dimensions, the system makes sure that the user has not inserted a negative number or zero. Also the system checks if the text field has been left empty. This is implemented as the following pseudo code describes in the table 5.5.2.

**Table 5.5.2**

| If ( ( string_rows==null ) || ( string_rows.length()==0 ) ) |
| { ... } |
| If ( ( rows==0) || ( rows<0 ) ) |
| { ... } |

The pseudo code in the above table describes how the system understands the wrong input when given from the users concerning the rows. In the same way, the system has similar methods for understanding when the number of columns given by the user is not appropriate for any game matrix.

The system also has a method for checking that the elements of the game matrix that the user is going to solve are properly given. Since zero elements are allowed as well as negative numbers, the system does not need to check in a way that it will be checking for the dimensions. It does, however, need to check that the user has actually inserted something, and moreover something numerical. For the former, this is implemented in the way illustrated by the black arrow in pseudo code in table 5.5.3 below. For the latter, this implemented in the way that the red arrow illustrates in the same table. These truth-checking methods that are described here are used throughout the system.

**Table 5.5.3**

For ( int i=0; i<matrix.length; i++)
{
    For ( int j=0; j<matrix[i].length; j++)
    {
        String Aij =JOptionPane.showMessageDialog("Enter Aij");
        → If ( ( Aij==null ) || ( Aij.length( ) == 0 ) )
        Try{double Aij=Double.parseDouble(Aij);  } Catch( Exception e){} 
    }
}
5.6 Display functions

The system has various ways for displaying the necessary output to the user. For different staff different operations are used. The system mainly uses popup screens and the Eclipse console.

For the Eclipse console the command “System.out.println” is used. Where the array, that is the game matrix, has to be displayed, the use of two nested loops will be necessary. This is done in a why that for every row element the column of that element is displayed.

To display the popup screens, the combination of “System.out.println” command together with the help of the “JOptionPane.showMessageDialog” command is used. Furthermore, the system holds in its memory all digits of a double number, but to the user the system only displays two digits. This is implemented with the help of the following command:

```
"DecimalFormatprecisionTwo=new DecimalFormat("0.00");"
```

For example, to display the game matrix that the user has inserted the program does it in a similar way that the pseudo code describes in the table 5.6 below.

**Table 5.6**

```java
String output = "\n";
stop:
{
   // matrix block
   for(int i=0; i<matrix.length; i++)
   {
      for (int j=0; j<matrix[i].length;j++)
      {
         if(i==matrix.length)
            break stop;
         double output2=0;
         output2 = matrix[i][j];
         output+="   "+output2;
      }
      output += "\n";
   }
   output += "\nThis is the matrix inserted";
}// End of matrix block
JOptionPane.showMessageDialog(null,output,"TheGameMatrix",
JOptionPane.INFORMATION_MESSAGE);
```


5.7 Minimax Maximin method

This fundamental theorem of game theory, which was given and proved by Von Neumann in 1928, is implemented in the following way. First the function minimax takes place. This function searches all the array’s columns one by one, and it holds in memory the maximum elements from each column. Then it stores in another variable the minimum element among those maximums.

Then the maximin function takes place. This function searches all the array’s rows one by one, and it holds in memory all the minimum elements from each row. Then it stores in another variable the maximum element among those minimums.

After both functions are finished, the system checks whether the two variables are the same, or with other words, if the two numbers belong to the same element. If that is the case, and minimax is equal with maximin, then the system has found a saddle point, the value of that element that both functions returned and the system is able to give the solution. If this function fails to find a saddle point, then the system passes the flow of control to the next function, the simplex method function.

For the implementation of the function, maximin, two nested loops are enough for going through all the elements of the matrix, which is basically going through all the numbers that are stored in the 2D array. This is because in Java this is done easily with the “matrix.length” command for getting the array’s 1st dimension first, and then for each recursion the command “matrix[i].length” is used for getting the 2nd dimension of the array. So in our case, for every element of the first row we are able to manipulate all the numbers below it, and that comprises all the rows of the game matrix. So for every row, as we go along the array, a temporary variable is used for storing the minimum element for that particular row. Then, when moving to the next row, if the minimum element of the current row is bigger than the variable that was stored before in the previous row, the system stores the new number to that variable. So when the system goes through the whole array, the value that is last stored in that variable is the maximum from all the minimum elements from all the rows, and in this way the system finds a value for maximin. The table 5.7.1 below describes with the help of pseudo code how this is implemented.

Table 5.7.1

```java
for(int i=0; i<matrix.length; i++)
{   min = matrix[i][0];  //First row element for i row
    for ( int j=0; j<matrix[i].length; j++)
    {   if (min > matrix[i][j] )
        {   min=matrix[i][j];   }
    }
    if ( maxmin < min)    //The Maximum element from all the minimums
        {   maxmin=min;   }
}
```
For the implementation of the other function, minimax, the method described above is not efficient and will not work for the simple reason that in Java the arrays are always manipulated from the first dimension. Therefore, some changes have been made on the array. What has been done is first to get the transpose of the game matrix and store it in another array, a temporary array, as in the example below.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{bmatrix}
\quad \text{will become} \quad
\begin{bmatrix}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 9 \\
\end{bmatrix}
\]

So from the array number 1 that is shown below, the array number 2 will be created as the diagrams show:

<table>
<thead>
<tr>
<th>Array 1</th>
<th>Array 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 4 7</td>
</tr>
<tr>
<td>4 5 6</td>
<td>2 5 8</td>
</tr>
<tr>
<td>7 8 9</td>
<td>3 6 9</td>
</tr>
</tbody>
</table>

Then the method follows a similar path as the maximin method. First, two nested loops are used for going through all the elements of the matrix, which is basically going through all the numbers that are stored in the 2D array. This is achieved again with the use of the “matrix.length” command for getting the array’s 1st dimension, and then for each recursion the command “matrix[i].length” is used for getting the 2nd dimension of the array. So in this case, for every element of the first row we are able to manipulate all the numbers below it, and that comprises all the rows of the game matrix. So for every row, as we go along the array, a temporary variable is used for storing the maximum element for that particular row. Then, when moving to the next row, if the maximum element of the current row is less than the variable that was stored before in the previous row, the system stores the new number to that variable. So when the system goes through the whole array, the value that is last stored in that variable is the minimum from all the maximum elements from all the rows, and in this way the system finds a value for minimax.

The table 5.7.2 below describes with the help of pseudo code how the temporary array is implemented.

As stated earlier, the minimax is implemented similarly as the maximin, but not in the array that contains the original game matrix but in the temporary array that is created by transposing the original one. That lets the columns get manipulated whilst they are in a row form. The pseudo code that describes how the minimax is implemented can be seen at the table 5.7.3.
Table 5.7.2

```java
... double[][] temp_matrix = new double[columns][rows]; ... 
For ( int i=0; i<matrix.length; i++)
{   w=0;
    For ( int j=0; j<matrix[i].length; j++)
    {
      temp_matrix[w][q]= matrix[i][j];
      w++;
    }
    q++;
}
```

Table 5.7.3

```java
For ( int i=0; i<temp_matrix.length; i++)
{   max = temp_matrix[i][0]; //First column element for i row
    For ( int j=0; j<temp_matrix[i].length; j++)
    {
      IF( max < temp_matrix[i][j] )
      {   max = temp_matrix[i][j];
      }
    }
    IF(minmax > max)//Minimum element from all the maximums
    {   minmax=max;
    }
}
```

At this stage, both minimax and maximin functions each return a value. These values are the variables minmax and maxmin respectively. By then comparing these two variables, the system is able to find out if saddle points exist and therefore the system can provide the solution to the user. If the two are not equal the system passes the flow of control to the simplex method.

Table 5.7.4

```java
IF ( minmax = = maxmin )
{   Saddle points exist - Give answers to the user - Exit   }
ELSE
{   Start simplex method       }
```
5.8 Simplex method

When the minimax maximin function fails to find a saddle point, the system’s flow of control is then passed to the simplex method function. This function gets the array that represents the game matrix, and then it performs a number of steps, operations, so as to give back the value of the game and the optimal strategies for the two players. The theory behind this method and how this performs the various steps mathematically has been described fully in the previous section, the Design section. In what follows here is how the system is implemented in order to perform all the steps that are necessary for the system to be able to find the solutions.

The first step, operation, of the simplex function that is performed is to check if the game matrix consists of positive elements only, or both positive and negative elements. The system is implemented to do that by the use of two nested loops in order to go through all the elements of the array one by one, and a variable that is initialised to zero in order to be compared with every element within the game matrix array. If any of the elements within the array are less than the zero value variable, then that element is stored in that variable replacing the zero value. When the system has gone through all the elements of the array, the variable used contains the element with the smallest value of the game matrix. By then checking what that value is, the system is able to understand if negative elements exist and which element has the smallest value. If the variable’s value is zero, then the system continues to the next step leaving all the elements of the array as they are. If the variable contains a negative value, then the system subtracts that negative value from all the elements of the array. In this way, since the variable used contains the smallest value, a negative value, all the elements of the array become positive and that value stays stored in the system in order to be added in the end. The pseudo code in the table 5.8.1 below, describes how this first operation is implemented.

Table 5.8.1

<table>
<thead>
<tr>
<th>check_for_negative=0;</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR ( int i=0; i&lt;matrix.length; i++)</td>
</tr>
<tr>
<td>{ FOR ( int j=0; j&lt;matrix[i].length; j++)</td>
</tr>
<tr>
<td>{ IF ( matrix[i][j] &lt; check_for_negative )</td>
</tr>
<tr>
<td>{ check_for_negative=matrix[i][j]; } }</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>IF(check_for_negative&lt;0)</td>
</tr>
<tr>
<td>{ FOR( int i=0; i&lt;matrix.length; i++)</td>
</tr>
<tr>
<td>{ FOR ( int j=0; j&lt;matrix[i].length; j++)</td>
</tr>
<tr>
<td>{ matrix[i][j]=matrix[i][j] - check_for_negative; } }</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
The next step, after making the array positive, was to create another array for storing and manipulating the semantic tableaus. This new array has one more extra row and one more extra column. The reason for that is explained in the Design section. This was achieved by using the command “double[] [] new_array = new double [rows+1][columns+1]”. Then this new array was filled using two nested loops and the original array. To the extra column the values of “1” was inserted and to the extra row the values of “-1”. For the extra row and column, the use of two nested loops was not necessary, so they were filled only by a single loop. For the last element of this new array, which is the $A_{\text{rows+1,columns+1}}$ element, the value of zero was inserted. For this value insertion, the use of any loop was not needed. The command “new_array[rows][columns]=0;” was efficient. The variables rows and columns have been used because arrays in Java start from zero and end up in one number less from the one they were created, so the numbers that the variables rows and columns represent, are the suitable ones.

Furthermore, inside the two nested loops, some local variables have been needed in order to count what is being filled at every loop and so as to insert the new elements when the time is appropriate.

The pseudo code in the table 5.8.2 below shows the way the new array is filled with the help of the local variables.

Table 5.8.2

```java
int ii, jj = 0;
...
FOR ( int i=0; i<matrix.length; i++ )
{  jj=0;
    FOR ( int j=0; j<matrix[i].length; j++)
    {  
      new_array[ii][jj]= matrix[i][j];
      jj++;
    } ii++;
}  
```

For creating the extra bottom row and for the insertion of the last element, the pseudo code looks like the table 5.8.3 below. In a similar way as for the extra row, the extra column was created.

Table 5.8.3

```java
FOR ( int j=0; j < columns; j++ )
{   new_array[rows][j] = -1 ;   }
...
...
new_array[rows][columns]=0;
```
The next step that the system does is to choose a pivot. Before the system chooses a pivot, it first checks that the bottom row includes at least one element with a negative value. The reason for doing this is explained in the description of the algorithm in the Design section. So from this point onwards the system recursively does the following operations until no elements have a negative value in the bottom row. The element where the value of the game is placed is excluded from this checking. The system then proceeds with the help of nested loops. It gets the elements of the new array and with the criteria for pivot selection it decides which element is appropriate for being the pivot. This is implemented by nested loops for criterion after criterion. The system by reaching the final loop for the pivots conditions, stores this element in a variable called pivot and then with the help of nested loops again inside the original recursion and after the final pivot selection loop, it then performs the operation that is called pivot step by the help of the variable that the pivot is stored. After all the elements have been changed and replaced with what is appropriate the system interchanges the “X” and “Y” labels that respect to the pivot in order to keep track on the final solutions. These labels are stored in two different one-dimensional arrays. Where necessary, local variables are used for keeping track of all the arrays as well as in all the nested loops. Once the system has finished the first loop of these operations, which is it does all the operations once, it performs a check to see whether the elements of the bottom rows are all positives. If that is the case, it then ends the recursion and proceeds to the final step, gets the solutions from the final tableau and then gives the solutions to the user. If a negative element still exists, then the system re-does all the operations from the beginning. The overall structure that has been described here can be seen in the table below:

```java
DO
    do outer loop:
        WHILE ( for_pivot != false)
            { 
                FOR( int i=0; i < new_array.length; i++)
                    {
                        FOR(int j=0; j < new_array[i].length; j++)
                            {
                                1st condition for selecting a pivot
                                    {
                                        2nd condition
                                            { 
                                                3rd condition
                                                    }   //end of 3rd condition
                                                for_pivot=false;
                                                break outer loop;
                                            }   //end of 2nd condition
                                        }   //end of 1st condition
                                    }   //end j loop
                                }   //end i loop:
                        } //end for_pivot
              } //end outer loop

Pivot Step
- change and replace all elements of the game matrix

Labels Step
- interchange x and y labels

System will check if negative values remain in the bottom row 
- IF Yes then it sets "Recuse-true;"
} WHILE(!recurse);
```
For the pivot selection, the table 5.8.4 describes with the help of pseudo code how the three criteria are checked.

Table 5.8.4

```
... IF( new_array[pivot_i][pivot_j] > 0 ) //1st condition
{  ...
    IF ( new_array[rows][pivot_j] < 0 ) //2nd condition
    {  ...
        IF ( smaller_positive_ratio > temp_ratio && temp_ratio>0) // 3rd Condition
        {  ...
            ...
            break outerloop;
        }
    }
}
```

For the pivot step process, the table 5.8.5 describes with pseudo code how this is implemented.

Table 5.8.5

```
FOR ( int i=0; i < new_array.length; i++) {  
    FOR ( int j=0; j<new_array[i].length; j++) {  
        //Elements that do not belong to neither pivot’s row or column
        //Element=element-(element(in pivot’s row)*element(in pivot’s column))/pivot
        IF( i  !=  pivot_i  &&  j  !=  pivot_j )  
        {  new_array[i][j]=new_array[i][j]-(([pivot_i][j]*new_array[i][pivot_j])/pivot);  }

        //Elements that belong to pivot’s row
        IF( i  ==  pivot_i  &&  j  !=  pivot_j )  
        {  new_array[i][j]=new_array[i][j]/ pivot ;  }

        //For elements that belong to pivot’s column
        IF( i  != pivot_i  &&  j  ==  pivot_j )  
        {  new_array[i][j]= -new_array[i][j] / pivot ;  }

        //For the pivot to be substitute with its reciprocal
        IF ( i  ==  pivot_i  &&  j  ==  pivot_j )  
        {  new_array[i][j]= 1 / new_array[i][j] ;  }
    }
}
```
Once the system breaks from the recursion, it gets the solutions from the final tableau and displays them to the user. This is done in the way the pseudo code describes in table 5.8.6. The “left_col_labels” and the “hor_row_labels” arrays seen in the table below are the two one-dimensional arrays that were used from the system to store the solutions, optimal strategies for the two players, in the right order.

Table 5.8.6

```
...  
...  
double value_of_the_game = 1 / new_array[rows][columns] ;
value_of_the_game = value_of_the_game + check_for_negative ;
...
...
FOR ( int I=0; I<left_col_labels.length; I++)
    { System.out.print( left_col_labels[I] +" " ) ; }  
...
...
FOR ( int I=0; I<hor_row_labels.length; I++)
    { System.out.print( hor_row_labels[I] +" " ) ; } 
...
...
System.out.println(“Optimal Value of the game:”+
                            precisionTwo.format( value_of_the_game ) );
...```
This chapter provides the reader with a detail explanation about the methods used for the testing purpose of the system. That includes methods descriptions and explanations on how these methods were applied…
6 Testing and Evaluation

The testing process is an essential stage of any software product since it ensures the correctness of a system. During this process any errors that might exist and rest in the system together with any miss functionality of the system can be discovered, recognised and therefore fixed and rectified. Additional to that, the system will become more reliable and anything that will force the system to have an unexpected behaviour can be discovered and corrected. Since the system is aimed at people that are more concerned in the mathematical way of solving game theory problems, it must be ensured that all the functions that the system provides run correctly without any fault and without giving any errors and moreover all the functions return the correct result.

The system was developed according to the waterfall model approach, as described in the Design chapter, and that consequently led to the development of the system by developing a number of incremental steps for all the methods and functions developed in the top down approach. That made possible for each unit to be tested whilst developing, and errors were discovered early and throughout the whole development process of the system. Moreover, in that way, it was ensured that errors would not become more complicated when the system was going to be fully developed, in a nested error hierarchy. In conclusion to that, the system was able to verify all the requirements and the functionality that was desired during development.

Testing phases Figure 23

After the completion of the first prototype, several techniques were used in order to test and evaluate the whole system. These techniques were mainly the verification and validation process, the defect testing technique, which included a black box testing technique and a structural testing technique, and the integration testing technique, that involved a combination of both top-down testing and bottom-up testing techniques. The errors that were identified, together with any unexpected behaviour that the system had during the above techniques, were fixed and the system was again retested until full and correct functionality was established.

6.1 Unit testing

The unit testing that took place during the development of the system ensured that all routines had the correct behaviour and consequently that they were returning the correct and desired output. The routines and methods that were involved are grouped into units and they are as shown in the diagram 6.1 below in the exact order that they were developed and tested.

23 Ian Sommerville, ‘Software Engineering, 6th edition’, p.441
The first unit that was developed was the mode selection unit. The mode selection unit was tested in the following way. First, by trying to select every one of the options among all the different ones that are available and then test that the system was performing the operation correctly. After it was concluded that the selected options were correctly stored in the system, the program was also tested for the behaviour it would have in the case of entering an invalid selection.

The next unit that was developed and tested was the Data insertion unit. This unit included both matrix dimensions insertion and matrix elements insertion. This unit was tested by inserting in the system all kind of different matrices with all kind of different dimensions. The arrays holding the matrices were then tested to see whether they held the complete and correct matrix elements.

The Display data unit was tested after being developed by combining it with the data insertion unit. This was basically done by giving any matrix to it and then displaying it on the console by the display Data unit that was going to be used throughout the system.

For the testing of the minmax function, pen and paper was needed. Basically, as a matrix was given as input to the system, that same matrix was solved parallely on paper for the existence of saddle points as well as which element of the matrix was the saddle point. This operation was done several times until the system was giving the same results for all different kind of matrices.

Similarly, for the testing of the simplex method function, paper and pen were used as well to find the value of the game and the optimal strategies for the two players. Also, examples from books were used in order to compare the system solutions with book solutions.

After all the above units were developed, tested, corrected and retested, the last unit was developed and tested. That was the Display results to the user unit. This unit was tested with the help of the eclipse console, and pop up screens. It combined all the above units and everything that the system had to provide to the user were checked that they were actually being provided. Further testing that was used can be found in the appendices.
6.2 Verification and Validation

Verification and validation (V&V) is a life-cycle process. V&V “is the name given to the checking and analysis process that ensure that software conforms to its specification and meets the needs of the customers…”

During the verification and validation process, two main techniques were used, the software inspections and the software testing. The figure 6.2.1 below shows how these two techniques were used in the different stages of development with the help of the arrows.

**Figure** 6.2.1

![Figure 6.2.1](image)

Furthermore, establishing the defects that existed in the system was the first step. The next step was to locate and correct these defects. That was archived by the debugging process. The debugging process steps and operations are shown in detail in the figure 6.2.3 below:

**Figure** 6.2.3

![Figure 6.2.3](image)

The questionnaires together with the evaluation conclusions that were used for the purpose of software inspections and software testing techniques can be found in the appendices.

---


6.3 Defect Testing

Defect testing was used for the system testing as well. It is a technique that is used to make the system fail in many various ways possible. It forces the system to perform incorrectly and therefore to expose any defect that might exist and rest in the system. This is exactly the opposite of the validation testing technique, which gave the system acceptance test cases and checked whether the system performed correctly. Both are needed in order to fully test and correct a system. The defect testing was performed in two ways, the black box testing and the structural testing since they are both required for a complete software examination.

6.3.1 Black Box Testing

Black box testing, or functional testing, as many people call it, is “an approach to testing where the tests are derived from the program or component specification. The system is a ‘black box’ whose behaviour can only be determined by studying its inputs and the related outputs”.[27]

Following this approach, the system has been given various inputs and the outputs were examined in detail to see whether these outputs corresponded to the given inputs. When the outputs were not the ones expected, then the test was considered to be successful, as the testing had revealed a problem within the system. The process was performed several times until no errors could be detected. The tests that were made can be found in the appendices.

6.3.2 Structural Testing

Structural testing approach is also called glass-box, clear-box, or even white-box testing to be distinguished from the black box testing. This is because in this approach “the tests are derived from knowledge of the software’s structure and implementation… The tester can analyse the code and use knowledge about the structure of a component to derive test data”.[28] This testing process was performed several times during the development process as well as after the completion and integration of the whole system. Since the author and the developer is the same person, no test cases were needed. The author knew exactly how to test all the several paths of the code since he knew the structure and how it was implemented. Additional to that, every routine and sub-routine was tested as soon as they were being developed.

6.4 Integration Testing

After individual components of the system were tested, they were integrated to make up the complete system. The integration tests that were made were drawn from the system specification and integration testing started as soon as usable components of

---

the system were put together, component after component. The testing involved in this section used a combination of top-down testing, starting from a high-level system and replacing individual components as it goes along, and bottom-up testing, which was integrating components in levels until the complete creation of the system.

Since the author and the developer is the same person, no test cases were needed. The author knew exactly how to test all the several units of the code after integration since he knew the structure of all units and perform the integration of the system himself.
Chapter 7

Conclusion

After having finished this dissertation, some conclusions were summarised together with some further additions and improvements. This is what this chapter describes.
7 Conclusion

The objective of this dissertation was to develop a program for solving two person zero sum games. The program produced is able to find optimal strategies for two players and the value of any game of two person zero sum type given. The program uses both minimax theorem, and the linear algebra with the simplex method in finding the solutions. The system produced meets the requirements described in chapter 3.

One of the main keys to a reliable and successful software system that always returns quick and correct solutions is solving according to a correct algorithm that is proved to be efficient and works. For this reason, the algorithm used for the purpose of this calculator was the simplex algorithm, a well-known algorithm. The top down approach to the design and the incremental development steps upon the system enabled the complexity of the simplex algorithm to be handled and has also resulted in a stable modular system that can be extended easily.

Although the system conformed to its requirements, some further improvements and additions could be made. These considerations could form part of any future work upon the system. The system was designed in a way that finds all the solutions algebraically. The system can be implemented to include a method that will be finding the solutions graphically, in order to give to the user a better understanding on maximization. Furthermore, the system can be extended in containing methods for solving games that are not of the zero sum type. Games like two person general sum or even n-person games. Additional to that, the system can include an option for teaching users that they have no experience on game theory how to solve a game matrix from the initial steps providing them with all the mathematics needed within the actual system. Furthermore, the system can also include a detail description within, for explaining and performing the simplex method to users with no previous experience in solving linear programming problems and consequently use the system as a tutor on the subject.

The user interface of the system provides the user with the basic functionality. Even though the user interface was not one of the primary concerns, it can be improved since it plays a big role in the system’s interaction with the users. It can give the ability to the user to insert different game matrices without having to exit the system and start all over again. Graphical comparisons between different game matrices can be provided to the user as well.

Having achieved the goal of implementing a calculator for solving two person zero sum games, the system was successful. Moreover, the system provides a function within that allows users to see the step-by-step calculations that are performed by the simplex algorithm, which can be helpful in other areas of mathematics.
Bibliography


Tucker, A. W., Luce, R. D., 1959. *Contribution to the theory of games*


Jacobson, Ivar., Object-Oriented Software Engineering: A use case Approach (ACM Press S.)


APPENDIX - A
APPENDIX A

A1 Evaluation of existing systems

Two-Person Zero-Sum and The Gambler’s Games with Applications
From: Professor Hossein Arsham

This system works by inserting the numbers of the matrix that the user desires in the white slots provided inside the purple box. Although this system gives the correct solutions, and the probabilistic numbers are correct, they answers were swapped and not in the correct place. First it gives the optimal strategy for player II but that it’s just an interface selection. The numbers that were misplaced are 0.75 and 0.25, the optimal strategy for Player I. Although to the eye does not make any difference, in mathematics that means either the algorithm is slightly wrong or changed, or either a completely different algorithm was used for finding the solutions which might not be as efficient and fast as solving it with linear programming and the simplex method. The optimal value of the game, that is 1.5, is correct.

The interface is really good though. It is simple, easy to use and straightforward. It does not confuse the user and everything the user needs is there in a logical place leaving the user no choice for mistakes or confusion.
The above program for solving two-person zero-sum games is accurate and the solutions are all correct but for the purpose of solving a game matrix is a little bit difficult to read. The answers are displayed in a graphical way, which at a first glance makes the user discontented. The program does not uses linear programming for solving the problem but it uses the graphical way or as a lot of people nowadays call it “the envelope method” because of the shape that gets created from the lines that intersect and looks like an envelope.

Also this program has too much unnecessary interface that makes it difficult for the user to use it quickly and efficiently.
A2 Questionnaire for potential users

Please fill the questions below with a black or blue pen

1. If you are currently studying, what is it that you do? If not, what is it that you studied?

2. Are you familiar with game theory?

3. Are you familiar with linear programming and the simplex method?

If your answers were “No” in questions 2 and 3, proceed to question number 8.

4. Have you ever solved a problem with linear programming and the simplex method?

5. Would you be interested in having a program for solving the two person zero sum games for you?

6. What exactly do you want from the program to do?

7. Is it something in particular that you want the program not to do?

If you have answered questions 4, 5, 6 and 7 then you are done.

8. Are you interested in learning what game theory is about?

9. What do you think is the best way of learning about this?

Thanks a lot for your time.
## A3 Use Cases

### 1. User runs the program

<table>
<thead>
<tr>
<th>Title</th>
<th>Program starting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>User, System</td>
</tr>
<tr>
<td>Precondition</td>
<td>Installation file was copied somewhere on the computer</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To start the calculator program</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System Primary Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>User runs the program file</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program started</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program failed to start</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>

### 2. Program mode

<table>
<thead>
<tr>
<th>Title</th>
<th>Mode selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>Program started and running</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To ask the user whether feedback should be provided or just quick solutions</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System Primary Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>User started the program</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>The options have been displayed</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program fails to display the options available to the user</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>
### 3. User selection

<table>
<thead>
<tr>
<th>Title</th>
<th>User chooses mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>User, System</td>
</tr>
<tr>
<td>Precondition</td>
<td>Program displayed the options available</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>For the user to select what mode the program will follow. Feedback or just solutions</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user started the program</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program accepted users selection and proceeds in the mode selected</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program fails to pass in the desired mode</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>

### 4. Matrix Row Dimension

<table>
<thead>
<tr>
<th>Title</th>
<th>Row Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>User, System</td>
</tr>
<tr>
<td>Precondition</td>
<td>User has selected the desired mode</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To get the matrix row dimension from the user</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user selected the mode and program continued</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>User gives the row dimension of the matrix and program accepts it</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program failed to accept the row dimension of the matrix</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>
### 5. Matrix column dimension

<table>
<thead>
<tr>
<th>Title</th>
<th>Row Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>User, System</td>
</tr>
<tr>
<td>Precondition</td>
<td>User has input row dimension and system has accepted it</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To get the matrix column dimension from the user</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user pressed enter after inserting row dimension for the matrix</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>User gives the column dimension of the matrix and program accepts it</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program fails to accept the column dimension of the matrix</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>

### 6. Display Dimensions of the game matrix

<table>
<thead>
<tr>
<th>Title</th>
<th>Matrix dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>User has input row and column dimensions of the matrix game</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To display the dimension of the matrix to the user</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user inserted column dimension and pressed enter</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>The dimension of the matrix are displayed to the user</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Dimension failed to be displayed</td>
</tr>
<tr>
<td>Priority</td>
<td>Low</td>
</tr>
</tbody>
</table>
### 7. Filling all the elements of the matrix

<table>
<thead>
<tr>
<th>Title</th>
<th>Filling the matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>User, System</td>
</tr>
<tr>
<td>Precondition</td>
<td>Program accepted and displayed the dimensions of the desired matrix</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>For the user to fill the matrix with elements, numbers of the game</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user pressed continue at the display dimension screen</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program accepted users inputs and stores all the elements of the game matrix in a two dimensional array</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program fails to get the values of the elements of the array from the user</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>

### 8. Confirmation and display

<table>
<thead>
<tr>
<th>Title</th>
<th>Confirmation and display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, user</td>
</tr>
<tr>
<td>Precondition</td>
<td>User has filled the matrix with all the elements and program accepted the values</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To show to the user what is going on at the present time</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user entered the last element of the array and pressed enter</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>System displays to the user a matrix of the dimension entered and filled with the elements entered</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program failed to display the matrix that the user has entered</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
</tbody>
</table>
### 9. System checks for saddle points

<table>
<thead>
<tr>
<th>Title</th>
<th>Saddle point checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>Program displayed the game matrix filled with the desired elements to the user</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To check the matrix and find a saddle point. An element where maxmin is equal with minmax</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user have pressed the continue button after the game matrix was displayed</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program will find a saddle point</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program will fail to find a saddle point</td>
</tr>
<tr>
<td>Priority</td>
<td>Low</td>
</tr>
</tbody>
</table>

### 10. When saddle points exist

<table>
<thead>
<tr>
<th>Title</th>
<th>Saddle points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>User has pressed the continue button after the game matrix was displayed and program started looking for saddle points</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>Display to the user which element of the game matrix is the saddle point and what the value of that element is. Value of the game is now found</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user selected the continue button after game matrix was displayed</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program displayed to the user the saddle point, value of the game, and exits</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program failed to display to the user the results</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>
### 11. When saddle points do not exist

<table>
<thead>
<tr>
<th>Title</th>
<th>No saddle points found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>Program has checked for saddle points and was not able to find any</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>The system will understand that saddle points do not exist and try to find the solutions with the simplex method</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user selected the continue button after game matrix was displayed</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program understood that saddle points do not exist and will perform another method for finding the solutions</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program fails to pass in the desired mode</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>

### 12. Another method for finding solutions

<table>
<thead>
<tr>
<th>Title</th>
<th>Simplex method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>System has understood that saddle points do not exist and an alternative method has started for finding the solutions</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To solve the game matrix with simplex method and find the optimal strategies for the two players and the value of this matrix game</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user selected the continue button after game matrix was displayed</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program performed simplex method and managed to find the optimal strategies for the two players and the value of the game</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program failed to perform simplex method</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>
### 13. Simplex method and solutions

<table>
<thead>
<tr>
<th>Title</th>
<th>Final solutions after simplex method had been performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>Program has used the simplex method in order to solve the game matrix</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>The system will display to the user the final value of the game and the optimal strategies for the two players</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user selected the continue button after game matrix was displayed</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program will display the value of the game and the optimal strategies for the two players to the user</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program fails to display the value of the game as well as the optimal strategies to the user</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
</tbody>
</table>

### 14. Quick mode

<table>
<thead>
<tr>
<th>Title</th>
<th>Quick mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>When the user had the option to select a mode, the feedback mode was selected</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>The system must perform all the operations quick and give the user quick results. Nothing else will be displayed. Only the answers of the game matrix</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System additional Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user selected the quick mode and pressed enter</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program understood that quick mode was selected and will give the user quick solutions. Only solutions will be displayed.</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program fails understand that quick mode was selected</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
</tbody>
</table>
### 15. Feedback mode

<table>
<thead>
<tr>
<th>Title</th>
<th>Feedback mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>System, User</td>
</tr>
<tr>
<td>Precondition</td>
<td>When the user had the option to select a mode, the feedback mode was selected</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>To give the user a step by step description of how the game was solved and results in each step as well as show all the semantic tableau's that were used and how these tableau's were modified</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System additional Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user selected the feedback mode and pressed enter</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>Program understood that feedback mode was selected and will give a description of all the steps that made in finding the solutions. Also additional information for pivoting and the semantic tableau's that were used from the creation of the first one to the final one will be displayed</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program will fail to give the user feedback</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
</tbody>
</table>

### 16. User wants to exit

<table>
<thead>
<tr>
<th>Title</th>
<th>Program closing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Secondary Actors</td>
<td>User, System</td>
</tr>
<tr>
<td>Precondition</td>
<td>Program is running</td>
</tr>
<tr>
<td>Goal in Context</td>
<td>User will press the “x” button in the top right corner in order to exit from the program</td>
</tr>
<tr>
<td>Scope and level</td>
<td>System main Task</td>
</tr>
<tr>
<td>Triggers</td>
<td>The user opened the program</td>
</tr>
<tr>
<td>Success End Condition</td>
<td>User exits and program closes</td>
</tr>
<tr>
<td>Failed End Condition</td>
<td>Program fails to close</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
</tbody>
</table>
APPENDIX - B
APPENDIX B

B1 Inspection and Evaluation testing

Two-Person Zero-Sum Calculator Questionnaire

Please run and try the two-person zero-sum calculator and then answer the questions in section A.

Section A

1. Did you find it easy to start the program?

2. Was it easy to follow the instructions given by the system? Were the instructions clear and concise?

3. Did you find it easy to insert the matrix game? If not, why?

4. Were you satisfied with the solutions given by the system?

5. Any suggestions and recommendations?

Thanks for your time.

Section B

Please do not fill this section in. It is for the use of the development team

Conclusions and Remarks
## B2  Defect Testing

<table>
<thead>
<tr>
<th>Requirement being tested</th>
<th>Input</th>
<th>Output</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user must be able to put any matrix dimension, m×n, that it is desired</td>
<td>A 200x200 matrix</td>
<td>Data Accepted</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must understand when the user did not put an integer number for the row dimensions of the matrix</td>
<td>A floating number for the row dimension</td>
<td>System identified that</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must understand when the user did not put an integer number for the column dimensions of the matrix</td>
<td>A floating number for the column dimension</td>
<td>System identified that</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must understand when a zero dimension was given as input for the rows of the matrix</td>
<td>Zero for the row dimensions</td>
<td>System identified that</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must understand when a zero dimension was given as input for the columns of the matrix</td>
<td>Zero for the columns dimensions</td>
<td>System identified that</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must understand when the user did not put numbers for the row and column dimensions of the matrix</td>
<td>Strings for the dimensions</td>
<td>System identified that</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system should check that the user did not leave any of the fields blank</td>
<td>Fields left blank</td>
<td>System identified that</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system should provide the user a facility for selecting between different modes</td>
<td>N/A</td>
<td>Different system modes provided</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system should provide a mode for providing feedback</td>
<td>N/A</td>
<td>Feedback mode is provided</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system should provide a mode for quick results</td>
<td>N/A</td>
<td>Quick mode is provided</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must understand when user has given a not numeric value for the elements of the matrix</td>
<td>A string for input</td>
<td>System identified that</td>
<td>Test Successful</td>
</tr>
<tr>
<td>Requirement being tested</td>
<td>Input</td>
<td>Output</td>
<td>Result</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>The system should be able to show all tableaus, pivots and pivoting step at each step of the algorithm</td>
<td>A game matrix and feedback mode selected</td>
<td>Tableaus, pivots and pivoting step at each step of the algorithm</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must not fail when the user has selected an invalid option for the mode</td>
<td>Invalid option selected</td>
<td>System identified that and continued</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must not fail when the user has left the selection field blank</td>
<td>Selection field left blank</td>
<td>System identified that and continued</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The user should be able to enter any number for the elements insertion</td>
<td>The numbers –23.54, 0, 426.57</td>
<td>Data Accepted</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must display to the user the game matrix, dimensions and elements, which has been inserted before solving the game</td>
<td>A game matrix</td>
<td>Everything was displayed before solving</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must be able to check how the problem can be solved</td>
<td>A game matrix</td>
<td>System identified what method has to be used</td>
<td>Test Successful</td>
</tr>
<tr>
<td>System must be able to find a saddle point when saddle points exist</td>
<td>A game matrix with saddle points</td>
<td>System identified the saddle point</td>
<td>Test Successful</td>
</tr>
<tr>
<td>System must inform the user when minmax is equal with maximin</td>
<td>A game matrix with saddle points</td>
<td>System informed about the equality</td>
<td>Test Successful</td>
</tr>
<tr>
<td>System must inform the user when minmax is not equal with maximin, and the system will use linear algebra and the simplex method</td>
<td>A game matrix with no saddle points</td>
<td>System informed the user about the absence of saddle points and for the usage of the simplex method</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must be able to proceed to the method that is going to be used for solving the problem automatically</td>
<td>A game matrix</td>
<td>System proceed to the solving method</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must be able to find the value of any game</td>
<td>A game matrix</td>
<td>System found the game value</td>
<td>Test Successful</td>
</tr>
<tr>
<td>Requirement being tested</td>
<td>Input</td>
<td>Output</td>
<td>Result</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------------------</td>
<td>------------------------------</td>
<td>------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>The system must be able to display to the user the value of any given game</td>
<td>A game matrix</td>
<td>System display the value</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must be able to find the optimal strategies for the both players of any game of the two-person zero-sum type</td>
<td>A random two-person zero-sum game</td>
<td>Optimal strategies for the two players found</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must be able to display to the user the optimal strategies for the both players of any game of the two-person zero-sum type</td>
<td>A random two-person zero-sum game</td>
<td>Optimal strategies for the two players displayed</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must provide a way for the user to exit the program at any given time</td>
<td>N/A</td>
<td>System provides exit buttons during all stages</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system should work in a way that minimises the possibilities for errors from the user</td>
<td>N/A</td>
<td>System guides the user from the beginning to the end</td>
<td>Test Successful</td>
</tr>
<tr>
<td>System must interact with the user only when necessary</td>
<td>N/A</td>
<td>The choices provided are limited. Only necessary ones</td>
<td>Test Successful</td>
</tr>
<tr>
<td>System must be able to explain to the user what the error was in the case of an error</td>
<td>Zero dimensions given as input</td>
<td>System informed that zero dimensions not allowed</td>
<td>Test Successful</td>
</tr>
<tr>
<td>The system must guide the user in a straightforward manner</td>
<td>N/A</td>
<td>System guides the user in a straightforward manner</td>
<td>Test Successful</td>
</tr>
</tbody>
</table>
B3 Book Testing

The following example is taken from the book *Game Theory*. Written by Guillermo Owen at page 54.

Solve the matrix game:

\[
\begin{pmatrix}
3 & 6 & 1 & 4 \\
5 & 2 & 4 & 2 \\
1 & 4 & 3 & 5
\end{pmatrix}
\]

Then he writes the maximizing problem in the form:

Maximize $\lambda$ subject to:

\[
\begin{align*}
3x_1 + 5x_2 + x_3 & \geq \lambda, \\
6x_1 + 2x_2 + 4x_3 & \geq \lambda, \\
x_1 + 4x_2 + 3x_3 & \geq \lambda, \\
4x_1 + 2x_2 + 5x_3 & \geq \lambda, \\
x_1 + x_2 + x_3 &= 1, \\
x_1, x_2, x_3 & \geq 0.
\end{align*}
\]

The final solutions that Owen finds are as follows:

Optimal strategy for Player A is: \( \frac{1}{8}, \frac{1}{2}, \frac{3}{8} \).

Optimal strategy for Player B is: \( \frac{1}{12}, \frac{5}{12}, \frac{1}{2} \).

Value of the game is \( \frac{13}{4} \).

Solving the same game matrix with the system produced, we find the solutions:

Optimal strategy for Player A: \( X_1=0.50 \quad X_2=0.13 \quad X_3=0.38 \quad X_4=0 \)

Optimal strategy for Player B: \( Y_1=0.42 \quad Y_2=0.08 \quad Y_3=0.50 \)

Optimal Value of the game: \( 3.25 \)

Which are the same with Owens book solutions since:

\( 1/8=0.13, \quad 3/8=0.38, \quad 1/2=0.5 \) AND \( 1/12=0.08, \quad 5/12=0.42, \quad 1/2=0.50 \)

And \( 13/4 = 3.25 \)
An example of a game matrix with saddle points is the one below:

$$
\begin{pmatrix}
  5 & 1 & 3 \\
  3 & 2 & 4 \\
 -3 & 0 & 1 \\
\end{pmatrix}
$$

The saddle point is the element $A_{22}$ of this matrix with value 2.

This example is taken from the book *Game Theory*. Written by Guillermo Owen at page 11.

Solving this example with the system gives the exact result:

Another two examples are the 7x2 and 2x7 matrices with saddle points below:

$$
\begin{pmatrix}
  1 & 2 \\
  3 & 4 \\
  5 & 6 \\
  7 & 8 \\
  9 & 10 \\
 11 & 12 \\
 13 & 14 \\
\end{pmatrix}
$$

First one has the saddle point $A_{71}$ element of the matrix with the value 13.
Second one has the saddle point $A_{21}$ element of the matrix with the value 8.
The system correctly identified both matrices, and returned back the following results respectively:

Saddle points found
Element A71 of the matrix

Saddle points found
Element A21 of the matrix
APPENDIX - C
APPENDIX C

C1 Reference Manual

Getting started with the
“Two-Person Zero-Sum Calculator”

A program based on game theory and linear algebra

VERSION 1.0

REFERENCE MANUAL

92
Contents

Sections

1. Getting started with the calculator

2. Mode Selection

3. Feedback Mode
   3.1. When Saddle points exist
   3.2. When Saddle points do not exist

4. Quick Mode
   4.1. When Saddle points exist
   4.2. When Saddle points do not exist

5. Error Handling, Wrong Input and Cancellation

For support and further information you can contact one of the members of the developing team.

The development team:
Argyrides Solonas: cs1sa@bath.ac.uk
Section 1
Getting started with the Calculator

Welcome to the Two-Person Zero-Sum Calculator system reference manual. This manual will provide a full reference guide to anything you need to know for using the system.

The system can be found in the enclosed CD that was included within the document.

First, copy the content of the CD in a directory or folder in your working environment and then follow the instructions within the txt file inside the CD.

Once you have copied the file in a desired direction or folder in your workspace, the calculator is ready for running. Then, run the execution file that is indicated in the CD and the system will start.

As soon as the system starts, depending on the environment you are using, the following screen will appear:

Windows Environment:

Running under the Eclipse Platform. Eclipse is free tool platform that can be downloaded at: http://www.eclipse.org/
As it is shown, any operating system that you may select to use, it will not affect the functionality of the system. Steps and procedures are the same, so it is no need for having all kinds of different user manuals for all the operating systems that the system can operate under. In what follows, Windows environment is going to be used for explaining and demonstrating the system throughout in this reference manual, under the eclipse platform.
Section 2
Mode Selection

The first screen that will appear when the calculator starts running, is the Mode Selection page. This is a page where the user can select what mode the calculator will follow. There are two options available. The first option is the feedback option. This option is offered for people that want to check either their solutions on their process. That includes semantic tableaus and pivots, as well as pivoting steps. The second option available is the quick mode selection. Selecting this option, the calculator will proceed with a mode that will only give the final value of the game and the optimal strategies for the two players. This option is good for experience users that want quick and correct results.

The page also includes an exit button denoted by “x” in the top right corner as well as a “Cancel” button for exiting the program immediately. The page also contains an “OK” button for pressing it after you have input in the field provided the option that is desired.

The page looks like this:

For a feedback selection insert in the field provided a “Y” or “y” and then press the “OK” button like the in the screen above.

For a quick mode selection insert in the field provided an “N” or “n” and then press the “OK” button.
To exit just press the “Cancel” button or the red square “x” in the top right corner.
Section 3
Feedback Mode

3.1
After you have selected ‘Y’ for feedback in the initial page, the calculator will automatically pass in the feedback mode. The next page that is displayed is the one below:

This page now guides the user in defining the game matrix dimensions that is about to be solved.

First, the user has to fill in the field shown above the number for the row dimension, and then to press enter. In our example we defined the row dimension to be 2.

At this point the user can exit the program as well by pressing the “x” button in the top right corner or the “Cancel” button as well, which is located next to the “OK” button.
After the user has pressed enter or “OK”, system will hold the number of rows in memory and display to the user another screen for filling in the column dimensions.

The screen will look like this:

![Input Screen](image)

At this point the user can exit the program as well by pressing the “x” button in the top right corner or the “Cancel” button as well, which is located next to the “OK” button.

After filling the field for the column dimensions you may proceed by pressing the “OK” button. The next page after the “OK” button has been pressed will look like the one below:

![Confirmation Screen](image)

This is more like a confirmation screen to ensure the user that the correct dimensions have been passed in the calculator’s memory. From here the user can either press the “OK” button to continue, or the red “x” button for exiting.
After the user has pressed the enter button the system proceeds in the way is shown below:

It gives the user an input field for filling the game matrix. By inserting the first element and pressing the “OK” button, the program will continue filling the matrix until the user has reaches the last element of the matrix. Let’s insert for simplicity the 2x2 matrix \[
\begin{pmatrix}
1 & 2 \\
3 & 4
\end{pmatrix}
\].

At any time the user can select the red “x” button at the top right corner or the “Cancel” button, for exiting and cancelling the operation respectively.
After the user has filled the game matrix, the system will tell the user that everything is ready and all the values have passed correctly to the memory, with a screen like this:

![Message screen](image1)

Again the system provides the little red “x” button so the user can immediately exit the program if she wants.

By pressing the “OK” button, the system will provide the user a screen for displaying what the game matrix is, in order for the user to check that this is what she wants.

![Confirmation screen](image2)

Two options are available in this confirmation screen. The “OK” button, which it will make the calculator proceed. The “x” button, which will make the operation cancelled. Proceeding with the “OK” button, program finds the solution if possible. In the example we are using, because saddle points do exist, the program found the solution as it is shown in the screen below.
In the example of the 2x2 game matrix \( \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \), the solution is 3, the \( A_{2,1} \) element of the matrix.

By pressing “OK”, system will exit and the solutions are going to be displayed in the Console like the screen below:
3.2
Now, when saddle points do not exist, system will proceed with the linear algebra and the simplex method. Feedback for all the tableaus will be given as well as pivots and pivoting steps. Consider the example \( A = \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \) of a 2x2 game matrix. the system will proceed like this:

Instead of displaying the solution like it did when saddle points existed. This screen is explaining that saddle points do not exist and that maxmin is not equal with maxmin. Therefore, the process that has to be used now is the linear algebra and the simplex method.

Like before the little red button “x” in the top right corner will let the user exit the program immediately.

The “OK” button, when pressed is going to start the algorithm for solving the problem.

Once the “OK” button has been pressed, the system will solve the problem and give the user the results and feedback of the process used for solving it. Semantic tableaus, pivots, temporary ratios and other useful feedback is going to be given to the user. That is the value of the game as well as the optimal strategies for the two players.
The screen below shows how the system will display that to the user:

```
Starting...
Feedback node selected
Filling the array...
Array has been filled

Starting MINIMAX Theorem... Checking for saddle points...
Maximum element from all the rows minimum: 1.0
Minimum element from all the columns maximum: 1.0
Saddle points do not exist
SIMPLEX Method Started...

Constructing first tableau...

1.00 2.00 1.00
3.00 0.00 1.00
-1.00 -1.00 0.00

Ratio to compare: 1.00
Ratio to compare: 0.33

Smaller positive ratio is 0.33
So the pivot at this step is 1.00
Performing pivot step with 1.00 being the pivot
Pivot step will give the following tableau:

-0.17 0.50 0.53
0.33 -0.00 0.33
-0.17 0.50 0.67
```

The above is the final tableau. The optimal strategies for the two players and the value of the game can be now resolved.

Optimal strategy for Player A:
\[ x_1 = 0.25, x_2 = 0.75 \]

Optimal strategy for Player B:
\[ y_1 = 0.50, y_2 = 0.50 \]

Optimal Value of the game: 1.50
Section 4
Quick Mode

4.1
After you have selected ‘N’ for no feedback in the initial page, the calculator automatically passes in the quick mode. The next page that is displayed is the one below:

This page now guides the user in defining the game matrix dimensions that is about to be solved.

First, the user has to fill in the field shown above the number for the row dimension, and then to press enter. In our example we defined the row dimension to be 2.

At this point the user can exit the program as well by pressing the “x” button in the top right corner or the “Cancel” button as well, which is located next to the “OK” button.
After the user has pressed enter, system will hold the number of rows in memory and display to the user another screen for filling in the column dimensions.

The screen will look like this:

![Input Screen]

At this point the user can exit the program as well by pressing the “x” button in the top right corner or the “Cancel” button as well, which is located next to the “OK” button.

After filling the field for the column dimensions and, you may now press the “OK” button to continue. The next page after the “OK” button has been pressed will look like the one below:

![Confirmation Screen]

This is more like a confirmation screen to ensure the user that the correct dimensions have been passed in the calculator’s memory. From here the user can either press the “OK” button to continue, or the red “x” button for exiting.
After the user has pressed the enter button the system proceeds in the way is shown below:

It gives the user an input field for filling the game matrix. By inserting the first element and pressing the “OK” button, the program will continue filling the matrix until the user has reaches the last element of the matrix. Let’s insert for simplicity the 2x2 matrix $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

At any time the user can select the red “x” button at the top right corner or the “Cancel” button, for exiting and cancelling the operation respectively.
After the user has filled the game matrix, the system will tell the user that everything is ready and all the values have passed correctly to the memory, with a screen like this:

![Message dialog box](image1.png)

Again the system provides the little red “x” button so the user can immediately exit the program if she wants.

By pressing the “OK” button, the system will provide the user a screen for displaying what the game matrix is, in order for the user to check that this is what she wants.

![Game Matrix dialog box](image2.png)

Two options are available in this confirmation screen. The “OK” button, which it will make the calculator proceed. The “x” button, which will make the operation cancelled. Proceeding with the “OK” button, program finds the solution if possible. In the example we are using, because saddle points do exist, the program found the solution as it is shown in the pop-up below.

![Message dialog box](image3.png)

In the example of the 2x2 game matrix\( \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \), the solution is 3, the \( A_{2,1} \) element of the matrix.
By pressing “OK”, system will exit and the solutions are going to be displayed in the Console like the screen below:

4.2 And similarly as in with the feedback mode, when saddle points do not exist, system will proceed with the linear algebra and the simplex method. Considering the example

\[ A = \begin{pmatrix} 1 & 2 \\ 3 & 0 \end{pmatrix} \]

Of a 2x2 game matrix, the system will proceed like this:

Instead of displaying the solution like it did when saddle points existed. This screen is explaining that saddle points do not exist and that maxmin is not equal with maxmin.
Therefore, the process that has to be used now is the linear algebra and the simplex method.
Like before the little red button “x” in the top right corner will let the user exit the program immediately.

The “OK” button, when pressed is going to start the algorithm for solving the problem.

Once the “OK” button has been pressed, the system will solve the problem quick and give the user the results. That is the value of the game as well as the optimal strategies for the two players.

The screen bellows shows how the system will display that to the user:

```
<terminated> game_theory [Java Application] C:\Program Files\Java\jre1.4.2_05\bin\java.exe (35-Apr-2001)
Starting...
Quick mode selected
Optimal strategy for Player A:
X1=0.25  X2=0.75

Optimal strategy for Player B:
Y1=0.50  Y2=0.50

Optimal Value of the game:1.50
```
Section 5
Error Handling, Wrong Input and Cancellation

The system provides some facilities for retrieving from errors and wrong inputs. For example, in the mode selection stage if a user instead of selecting a proper selection does something that the system will not expect, the system will not fail.

Consider the example:

The system will not fail when the “OK” enter is pressed. It will actually proceed like the screen below:
When a user acts in a way that he is not expected, like defining a row or a column dimension with zero, or inserting invalid input when filling the elements, for example:

![Screenshot of input dialog]

The system will immediately understand that and act the way is shown in the screens below:

![Screenshot of error dialog]

Also when the user on purpose tries to proceed without feeling dimensions or, and elements of the matrix, system will understand it and act like the screen below:

![Screenshot of console output]

Invalid selection
For the row dimensions an integer has to be inserted
When the user wants to exit at any time, that is done by either the “Cancel” button or the little red button denoted by “x” at the top right corner of the screen.

For example in the first screen,

![First Screen Image]

When the user presses “x” or “Cancel” the system immediately exits

![Second Screen Image]
APPENDIX - D
import javax.swing.*;
import java.text.DecimalFormat;

public class game_theory {

    public static void main(String[] args) {

        // Small tests are used frequently to help at compile time and also
        // to keep on track of what the program is doing step by step. Also
        // in this way the user gets feedback for what is done behind the scenes

        System.out.println("Starting..."); // Testing 1

        // To ask the user whether he wants to be provided with feedback or not
        String Mode = JOptionPane.showInputDialog("This is a program for solving
        Two-Person " + 
        " Zero-Sum games. For a detailed step by step description and
        feedback press" + 
        "'Y'. \
        For solutions only press 'N'");

        // Input Validity selection check
        String Y="Y";
        String N="N";
        if((Mode==null)||(Mode.length()==0))
        {
            System.out.println("Operation is cancelled");
            System.exit(0);
            
        }
        else if( Mode.equalsIgnoreCase(Y))
        {
            System.out.println("Feedback mode selected");
        }
        else if(Mode.equalsIgnoreCase(N) )
        {
            System.out.println("Quick mode selected");
        }
        else
        {
            System.out.println("Invalid selection\n" +
            "Program will proceed in quick mode");
        }

        // To get the game matrix rows
        String input = JOptionPane.showInputDialog("Please enter the number of rows:");
        // Input Validity check
        if((input==null)||(input.length()==0))
try{
    int matrix_row_dimension = Integer.parseInt(input);
} catch(Exception ex) {
    System.out.println("Invalid selection");
    System.out.println("For the row dimensions an integer has to be inserted");
    String error="The input was invalid
Program will exit";
    JOptionPane.showMessageDialog(null,error,"ERROR",
    JOptionPane.ERROR_MESSAGE);
    System.exit(0);
}
int rows = Integer.parseInt(input);//To convert the input

//To get the game matrix columns
String input2 = JOptionPane.showInputDialog("Please enter the number of columns:");

//Input Validity check
if(input2==null||(input2.length()==0)) {
    System.out.println("Input not found\nOperation is cancelled");
    System.exit(0);  }
try{
    int matrix_column_dimension = Integer.parseInt(input2);
} catch(Exception ex) {
    System.out.println("Invalid selection");
    System.out.println("For the column dimensions an integer has to be inserted");
    String error="The input was invalid\nProgram will exit";
    JOptionPane.showMessageDialog(null,error,"ERROR",
    JOptionPane.ERROR_MESSAGE);
    System.exit(0);
}
int columns = Integer.parseInt(input2);

//Checks for input validity
if((rows==0)||(columns==0)||(rows<0)||(columns<0)) {
    String error="The input was invalid\nProgram will exit";
    JOptionPane.showMessageDialog(null,error,"ERROR",
    JOptionPane.ERROR_MESSAGE);
    System.exit(0);
}

//Displays to the user the matrix dimensions that had been given
JOptionPane.showMessageDialog(new JFrame(),"The matrix" +
    " you have given is going to be a "+rows+"x"+columns+
    " matrix! " + "Press enter to fill the matrix!");

int rows2=rows-1;      //Since arrays in java start at zero, these
int columns2=columns-1;//numbers are going to be used when necessary
//Creates the matrix of the dimensions given
double[][] matrix = new double[rows][columns];

DecimalFormat precisionTwo = new DecimalFormat("0.00");
//The precisionTwo is going to be used throughout the program,
//When tableaus are being displayed and also to the final results,
//To make them all in two digits to the right of the decimal point

if(Mode.equalsIgnoreCase(Y))
{System.out.println("Filling the array...");}//Testing 2

//To fill the array with the matrix elements
for (int i=0; i<matrix.length; i++)
{
    for(int j=0; j<matrix[i].length; j++)
    {
        //To keep track of the matrix elements
        int count=i+1;
        int count2=j+1;

        //To get element i,j of the matrix
        String input3=JOptionPane.showInputDialog("Please"+
        " enter the element A"+count+count2+":");

        //Checks for input validity
        if((input3==null)||(input3.length()==0))
        {
            System.out.println("Input not found
Operation is cancelled");
            System.exit(0);
        } try
        {
            double Aij=Double.parseDouble(input3);
            matrix[i][j]=Aij; //Places each element in the array
        }
        catch(Exception ex)
        {
            System.out.println("Invalid selection");
            System.out.println("Operation is canceled");
            String error="The input was invalid
Program will exit";
            JOptionPane.showMessageDialog(null, error ,"ERROR",
                JOptionPane.ERROR_MESSAGE );
            System.exit(0);
        }
    }
}
if(Mode.equalsIgnoreCase(Y))
{System.out.println("Array has been filled");}//Testing 3

JOptionPane.showMessageDialog(new JFrame(),"The Game" +
" Matrix is ready. Press enter to view");

//To display the elements of the matrix to the user
String output =""
stop: { // matrix block
    for(int i=0; i<matrix.length; i++) {
        for(int j=0; j<matrix[i].length; j++) {
            if(i==matrix.length)
                break stop;
            double output2=0;
            output2 = matrix[i][j];
            output+="      "+output2;
        }
        output += "\n";
    }
    output+="\nThis is the matrix inserted. Press enter to continue";
}// End of matrix block

JOptionPane.showMessageDialog(null, output , "The "+
"Game Matrix", JOptionPane.INFORMATION_MESSAGE);

if(Mode.equalsIgnoreCase(Y)){
    System.out.println("Matrix was displayed"); //Testing 4
    System.out.println("Starting MINIMAX Theorem... " +
    "Checking for saddle points... "); }//Testing 5

/* To check if saddle points exists. With Minimax Theorem
Checks all elements of all rows and takes all the minimums and then
the maximum among them. Then checks all columns and takes the maximums
from all and then from them the minimum among them. After that compares
"max min" for equality with "min max" */

double min=0;//Variable to use for comparing each element of a row
double maxmin=-999999;
int min_i=0;//Variable to keep track of the i position of the element
int min_j=0;//Variable to keep track of the j position of the element
int maxmin_i=0;
int maxmin_j=0;

for(int i=0; i<matrix.length; i++)
{
    min = matrix[i][0]; //First row element for i row
    min_i=i;
    min_j=0;

for (int j=0; j<matrix[i].length; j++)
{
    if (min > matrix[i][j] )
    {
        min=matrix[i][j];
        min_i = i;
        min_j= j;
    }
}

if( maxmin < min) //The Maximum element from all the minimums
{
    maxmin=min;
    maxmin_i= min_i + 1;
    maxmin_j= min_j + 1;
}

if(Mode.equalsIgnoreCase(Y))
{System.out.println("Maximum element from all the rows minimums: "+maxmin);}

    //A temporary matrix needs to be created here so as
    //to get the columns first instead of the rows
    int q=0; int w=0;
    double[][] temp_matrix = new double[columns][rows];

    for(int i=0; i<matrix.length; i++)
    {
        w=0;
        for(int j=0; j<matrix[i].length; j++)
        {
            temp_matrix[w][q]= matrix[i][j];
            w++;
        }
        q++;
    }

double max=0; //Variable to use for comparing each element of a column
double minmax=9999999;

for(int i=0; i<temp_matrix.length; i++)
{
    max = temp_matrix[i][0]; //First column element for i row
    for(int j=0; j<temp_matrix[i].length; j++)
    {
        if(max < temp_matrix[i][j] )
        {
            max = temp_matrix[i][j];
        }
    }
}

if( minmax > max) //The Minimum element from all the maximums
{
    minmax=max;
}
if(Mode.equalsIgnoreCase(Y))
{ System.out.println("Minimum element from all the columns maximums:
"+minmax+";\n\n\nif(minmax==maxmin)
{
 JOptionPane.showMessageDialog(new JFrame(),"The matrix " +
    "Game has the saddle point "+ minmax + " since the minimax " +
    " = maximin = "+ minmax +".\nThis is the element "+
    "A" + maxmin_i + maxmin_j + " " + "of the matrix. "+
    "This is the value of the game");

 System.out.println("Saddle points found");//Testing 6
 System.out.println("Element A"+ maxmin_i + maxmin_j +" of the matrix");
}
else
{
 JOptionPane.showMessageDialog(new JFrame(),"This "+
    "Matrix Game does not have a saddle point!\n" +
    "MinMax="+minmax+"\n\tMaxMin="+ maxmin +"\n"+
    "Minimax" + " is not equal with Maximin\n" +"\n"+
    "So the program has to proceed with linear algebra\n"+
    "It will use the simplex method to find the optimal\n"+
    "strategies for the two players and the value of the " +
    "game\n"+"\n" +
    "Press enter to continue");

 if(Mode.equalsIgnoreCase(Y))
{ System.out.println("Saddle points do not exist\nSIMPLEX Method Started...\n\n\n//Starting simplex method
//1st STEP
//Checks if there exists negative entries in the matrix game

double check_for_negative=0;
for(int i=0; i<matrix.length; i++)
{
 for(int j=0; j<matrix[i].length; j++)
{
    if ( matrix[i][j] < check_for_negative )
    {
        check_for_negative=matrix[i][j];
    }
}
}

//If there exists a negative entry in the matrix game, the
//program will add a number to make matrix entries positive.
if(check_for_negative<0)
{
    for(int i=0; i<matrix.length; i++)
    {
        for(int j=0; j<matrix[i].length; j++)
        {
            matrix[i][j]=matrix[i][j]-check_for_negative;
        }
    }
}

//2nd STEP
//Creates the tableau
int simpl_matr_rows = rows + 1; //for the rows of the tableau
int simpl_matr_cols = columns+1; //for the columns of the tableau

//For filling the tableau
int ii=0;
int jj=0;

//Array to be used for the first tableau of the simplex method
double[][] matrix_for_simplex=new double[simpl_matr_rows][simpl_matr_cols];

for(int i=0; i<matrix.length; i++)
{
    jj=0;
    for(int j=0; j<matrix[i].length; j++)
    {
        //To fill the tableau
        matrix_for_simplex[ii][jj]= matrix[i][j];
        jj++;
    }
    ii++;
}

for(int j=0; j<simpl_matr_cols-1; j++ ) //To fill the last Row
{
    matrix_for_simplex[simpl_matr_rows-1][j]=-1;
}

for(int i=0; i<simpl_matr_rows-1; i++ ) //To fill the last Column
{
    matrix_for_simplex[i][simpl_matr_cols-1]=1;
}

//To fill the element where the value of the game is going to be
matrix_for_simplex[simpl_matr_rows-1][simpl_matr_cols-1]=0;
if(Mode.equalsIgnoreCase(Y))
{System.out.println("nConstructing first tableau..\n");}//Testing 6

//To Print the first tableau
for(int i=0; i<matrix_for_simplex.length; i++)
{
    for(int j=0; j<matrix_for_simplex[i].length; j++)
    {
        //To print the initial tableau
        if(Mode.equalsIgnoreCase(Y))
        {System.out.print(" "+ precisionTwo.format(matrix_for_simplex[i][j])+"\t");}
    }
    if(Mode.equalsIgnoreCase(Y))
    {System.out.println("n");}
}

//3rd STEP
//Choses a pivot according to the 3 criteria for selecting pivots
//And then it makes pivot steps to get the next tableau
//Checks wether a negative entry remains in the bottom row of the tableau

//If a negative entry remains, it repeats the procedure

double temp_ratio=0;
double smaller_positive_ratio=1000000;
double pivot=0;
double[] pivot_col = new double[simpl_matr_rows];
double[] pivot_last_col = new double[simpl_matr_rows];
boolean for_pivot=true;
boolean recurse=false;
int pivot_i=0;
int pivot_j=0;
String[] x_labels = new String[simpl_matr_rows-1];
String[] y_labels = new String[simpl_matr_cols-1];
String[] left_col_labels=new String[simpl_matr_cols-1];
String[] hor_row_labels=new String[simpl_matr_rows-1];
int zero=0;
String s_zero="0";

int initialisation_x=1;
int initialisation_y=1;

//Labels are used for keeping track on players strategies
//To fill the x labels array
for(int x=0; x<x_labels.length; x++)
{
    x_labels[x]="X"+initialisation_x;
    initialisation_x++;
}
//To fill the y labels array
for(int y=0; y < y_labels.length; y++)
{
    y_labels[y]="Y"+initialisation_y;
    initialisation_y++; 
}

//Since negative entries exist, program will start the method
//After first loop, program will recurse if negative entries
//remain in the bottom row of the matrix

do { // START OF DO DO
    outerloop:
    while(for_pivot != false)
    {
        for(int inner_i=0; inner_i < matrix_for_simplex.length; inner_i++)
        {
            for(int inner_j=0; inner_j < matrix_for_simplex[inner_i].length; inner_j++)
            {
                //1st condition for selecting a pivot
                if( matrix_for_simplex[inner_i][inner_j] > 0 )
                {
                    //2nd condition for selecting a pivot
                    if(matrix_for_simplex[simpl_matr_rows-1][inner_j] < 0 )
                    {
                        //To fill the last column for ratio
                        for(int gg=0; gg<simpl_matr_rows-1; gg++)
                        {
                            pivot_last_col[gg] = matrix_for_simplex[gg][simpl_matr_cols-1];
                        }
                        //To fill pivot s column for ratio
                        for(int gg=0; gg<simpl_matr_rows-1; gg++)
                        {
                            pivot_col[gg] = matrix_for_simplex[gg][inner_j];
                        }
                        //To find temporary ratio
                        for(int piv=0; piv<pivot_last_col.length-1; piv++)
                        {
                            temp_ratio = pivot_last_col[piv]/pivot_col[piv];
                            if(Mode.equalsIgnoreCase(Y))
                            {
                                System.out.print("Ratio to compare:" + precisionTwo.format(temp_ratio) + "\n");
                            }
                        }
                    }
                }
            }
        }
    }
} // END OF DO
This also satisfies 3rd condition: Since smaller_positive_ratio is bigger than temp_ratio and temp_ratio is bigger than zero, that means smaller_positive_ratio is bigger than zero as well /*

if( smaller_positive_ratio > temp_ratio && temp_ratio > 0 )
{
    smaller_positive_ratio=temp_ratio;
    pivot=matrix_for_simplex[piv][inner_j];
    pivot_i=piv;
    pivot_j=inner_j;
}
} //end of 3rd condition

for_pivot=false;
break outerloop;

} //end of 2nd condition

} //end of 1st condition

} //end inner_j loop
} //end inner_i loop
} //boolean for pivot

if(Mode.equalsIgnoreCase(Y))
{
    System.out.print("\nSmaller positive ratio is "+ precisionTwo.format(smalle

r_positive_ratio) +"\n");
    System.out.print("So the pivot at this step is "+precisionTwo.format(pivot)+" \n");
    System.out.print("Performing pivot step with "+precisionTwo.format(pivot)+" being the pivot\nPivot step will give the following tableau:\n\n");
}

//4th STEP
//Pivot Step
//After selecting a pivot and checking for the criteria validity,
//program will perform the pivot step, and get the next tableau
//for consideration

//For elements that don t belong to pivots row or column
for(int piv_i=0; piv_i < matrix_for_simplex.length; piv_i++)
{
    for(int piv_j=0; piv_j<matrix_for_simplex[piv_i].length; piv_j++)
    {
        //Element=element-(element(in pivot`s row)*element(in pivot`s column))/pivot
        if(piv_i!=pivot_i && piv_j!=pivot_j)
        {
            matrix_for_simplex[piv_i][piv_j]=matrix_for_simplex[piv_i][piv_j]-
            ((matrix_for_simplex[pivot_i][piv_j]* matrix_for_simplex[piv_i][pivot_j])/pivot);
        }
    }
}
//For elements that belong to pivots row
for(int piv_i=0; piv_i < matrix_for_simplex.length; piv_i++)
{
    for(int piv_j=0; piv_j<matrix_for_simplex[piv_i].length; piv_j++)
    {
        //Element=element/pivot
        if(piv_i==pivot_i && piv_j!=pivot_j)
        {
            matrix_for_simplex[piv_i][piv_j]=matrix_for_simplex[piv_i][piv_j]/pivot;
        }
    }
}

//For elements that belong to pivots column
for(int piv_i=0; piv_i < matrix_for_simplex.length; piv_i++)
{
    for(int piv_j=0; piv_j<matrix_for_simplex[piv_i].length; piv_j++)
    {
        //Element= -element/pivot
        if(piv_i!=pivot_i && piv_j==pivot_j)
        {
            matrix_for_simplex[piv_i][piv_j]= -matrix_for_simplex[piv_i][piv_j]/pivot;
        }
    }
}

//For the pivot to be substitute with its reciprocal
for(int piv_i=0; piv_i < matrix_for_simplex.length; piv_i++)
{
    for(int piv_j=0; piv_j<matrix_for_simplex[piv_i].length; piv_j++)
    {
        //Element(which is the pivot here)=1/pivot
        if(piv_i==pivot_i && piv_j==pivot_j)
        {
            matrix_for_simplex[piv_i][piv_j]=1/matrix_for_simplex[piv_i][piv_j];
        }
    }
}

//For printing tableau at every stage
for(int piv_i=0; piv_i < matrix_for_simplex.length; piv_i++)
{
    for(int piv_j=0; piv_j<matrix_for_simplex[piv_i].length; piv_j++)
    {
        if(Mode.equalsIgnoreCase(Y))
{ System.out.print(" + precisionTwo.format  
(matrix_for_simplex[piv_i][piv_j])\"\t\"); }

if(Mode.equalsIgnoreCase(Y))
{ System.out.println("n"); }

//End of 4th Step - Pivot Step

//5th Step
//Now the the program will interchange x labels
//with y labels to keep truck on the final solutions

String interchange = x_labels[pivot_i];
String interchange2= y_labels[pivot_j];

x_labels[pivot_i]=interchange2;
y_labels[pivot_j]=interchange;

//End of 5th STEP

//6th STEP
//The program will check if negative values remain in the bottom row

int checking_for_neg = 1;

innermost:
for(int i=matrix_for_simplex.length-1; i<matrix_for_simplex.length; i++)
{
    for(int j=0; j<matrix_for_simplex[i].length-1; j++)
    {
        if(matrix_for_simplex[i][j] < 0)
        {
            checking_for_neg=-1;
        }
        //If bottom rows element are positive then
        //sets recurse to true and the loop ends
        //Values can be read off
        if(checking_for_neg == 1)
        {
            recurse=true;
        }
        //If bottom rows element are positive then
        //sets recurse to false and it loops again
        //and also initialises necessary variables
        else
        {
            recurse=false;
        }
    }
}

125
temp_ratio=0;
smaller_positive_ratio=1000000;
pivot=0;
for_pivot=true;
pivot_i=0;
pivot_j=0;

break innermost;
}
}

} while(!recurse); //END OF DO (3rd STEP) together with the 6th step
//It will loop until no negative entries remain in the bottom row
//Apart from the last entrie though which is the value of the game.

//7th Step(Final step)
//To get read off the solutions

if(Mode.equalsIgnoreCase(Y))
{
    System.out.println("The above is the final tableau. The optimal strategies for " +
"the two players and the value of the game can be now resolved\n");
}

double value_of_the_game;
double final_value_of_the_game;
value_of_the_game=1/matrix_for_simplex[simpl_matr_rows-1][simpl_matr_cols-1];
final_value_of_the_game = value_of_the_game + check_for_negative;

// End of simplex method

//Sorting labels
for(int x=0; x < x_labels.length; x++)
{
    String origin = x_labels[x].substring(0,1);
    String tail = x_labels[x].substring(1);
    int ord_tail = Integer.parseInt(tail);
    int ind_tail = ord_tail-1;
    String xx="X";

    if(origin.equals(xx))
    {
        hor_row_labels[ind_tail]="Y"+x_labels[x].substring(1)+"="+s_zero;
    }
    else
    {
        left_col_labels[ind_tail]="X"+ x_labels[x].substring(1)+"="+ precisionTwo.format
(matrix_for_simplex[simpl_matr_rows-1][ind_tail]/matrix_for_simplex
[simpl_matr_rows-1][simpl_matr_cols-1]);
    }
for(int y=0; y < y_labels.length; y++)
{
    String origin= y_labels[y].substring(0,1);
    String tail = y_labels[y].substring(1);
    int ord_tail = Integer.parseInt(tail);
    int ind_tail = ord_tail-1;
    String yy="Y";
    
    if(origin.equals(yy))
    {
        left_col_labels[ind_tail] = "X"+y_labels[y].substring(1)+"=\"s_zero;"
    }
    else
    {
        hor_row_labels[ind_tail] ="Y"+y_labels[y].substring(1)+"=\"+ precisionTwo.format(matrix_for_simplex[ind_tail][simpl_matr_cols-1]/matrix_for_simplex[simpl_matr_rows-1][simpl_matr_cols-1]);
    }
}

//Labels have been sorted
//Printing the optimal strategies for player A
System.out.println("Optimal strategy for Player A: ");
for(int i=0; i<left_col_labels.length; i++)
{
    System.out.print( left_col_labels[i] +" ");
}

System.out.println("\n");

//Printing the optimal strategies for player B
System.out.println("Optimal strategy for Player B: ");
for(int i=0; i<hor_row_labels.length; i++)
{
    System.out.print( hor_row_labels[i] + " ");
}

//Printing the final value of the game
System.out.println("\n\nOptimal Value of the game:"+precisionTwo.format(final_value_of_the_game));

}//end of else( Either minmax=maxmin or simplex method )

System.exit(0);